ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ НАУК

МАТЕМАТИЧЕСКИЕ СТРУКТУРЫ И МОДЕЛИРОВАНИЕ

Выпуск 20

Омск 2009

Журнал «**Математические структуры и моделирование**». – Омск, 2009. – Вып. 20. – 191 с.

ISBN

Редакционная коллегия

Главный редактор
А.К. Гуц
д-р физ.-мат. наук, профессор

Н.Ф. Богаченко
к-т физ.-мат. наук

Д.Н. Лавров
к-т техн. наук

Е.В. Палешева
к-т физ.-мат. наук

Художественное оформление
В.В. Коробицын

Адрес научной редакции

Россия, 644053, Омск - 53, ул. Грозненская, 11
Омский государственный университет
факультет компьютерных наук

E-mail: guts@omsu.ru
msm@cmm.univer.omsk.su

ISBN

© Омский госуниверситет, 2009

МАТЕМАТИЧЕСКИЕ СТРУКТУРЫ и МОДЕЛИРОВАНИЕ

В журнале публикуются статьи, в которых излагаются результаты исследований по фундаментальной и прикладной математике, теоретической физике и размышления, касающиеся окружающей нас природы и общества.

Публикуются также статьи по информационным технологиям, компьютерным наукам, защите информации, философии и истории математики.

Объекты исследования должны быть представлены в форме некоторых математических структур и моделей.

Журнал является реферируемым. Рефераты статей публикуются в «Реферативном журнале» и в журналах «Zentralblat für Mathematik» (Германия) и «Mathematical Reviews» (США).

Электронная версия журнала представлена в сети «Интернет» по адресу:

`http://cmm.univer.omsk.su`

Журнал издается на коммерческие средства факультета компьютерных наук Омского государственного университета.

Наш E-mail:

`msm@cmm.univer.omsk.su`

Подробную информацию можно найти на Web-сервере:

`http://cmm.univer.omsk.su`

СОДЕРЖАНИЕ

Фундаментальная математика

Прикладная математика и моделирование

Информационные технологии

продолжение на след.странице

Защита информации

Методика преподавания

Математические структуры и моделирование
2009, вып. 20, с. 5–11

УДК 519.214.5

ОБОБЩЕНИЕ НЕРАВЕНСТВА ИБРАГИМОВА

А.Г. Гринь

В работе получен аналог известного неравенства И.А. Ибрагимова для моментов слабо зависимых случайных величин, использующий нормы более общего вида, чем в оригинальном неравенстве – так называемые нормы Орлича.

Пусть $\{\xi_n\} = \{\xi_n,\ n = 1, 2, ...\}$ – стационарная в узком смысле последовательность и пусть $\mathcal{F}_{\le n}$ и $\mathcal{F}_{\ge n}-\sigma$ -алгебры, порожденные семействами $\{\xi_i : i \le n\}$ и $\{\xi_i : i \ge n\}$. Говорят, что последовательность $\{\xi_n\}$ удовлетворяет *условию равномерно сильного перемешивания (φ-перемешивания)* с коэффициентом перемешивания $\varphi(n)$, если

$$\varphi(n) = \sup\left\{\frac{|\mathbf{P}(AB) - \mathbf{P}(A)\mathbf{P}(B)|}{\mathbf{P}(A)} :\ A \in \mathcal{F}_{\le 0},\ B \in \mathcal{F}_{\ge n}\right\} \to 0, \quad n \to \infty.$$

Пусть последовательность $\{\xi_n\}$ удовлетворяет условию φ–перемешивания с коэффициентом перемешивания $\varphi(n)$ и пусть случайные величины ξ и η измеримы относительно $\mathcal{F}_{\le 0}$ и $\mathcal{F}_{\ge n}$ соответственно,

$$\|\xi\|_p = (\mathbf{E}|\xi|^p)^{\frac{1}{p}} < \infty, \quad \|\eta\|_q < \infty, \quad p > 1,\ q > 1, \quad p^{-1} + q^{-1} = 1.$$

Тогда

$$|\mathbf{E}\xi\eta - \mathbf{E}\xi\,\mathbf{E}\eta| \le 2\varphi^{\frac{1}{p}}(n)\|\xi\|_p\|\eta\|_q. \tag{1}$$

Доказательство см., например, в [1, теорема 17.2.3].

Неравенство (1), полученное И.А. Ибрагимовым, является одним из основных инструментов при доказательстве предельных теорем для сумм слабо зависимых случайных величин. Ниже доказано обобщение неравенства (1), использующее нормы более общего вида, чем $\|\xi\|_p$ – так называемые нормы Орлича.

Приводимые ниже определения и доказательство соотношений (2)-(6) можно найти в [2].

Непрерывную выпуклую функцию $f(x),\ x \in (0, +\infty)$ назовем N-функцией, если

$$\lim_{x\to+0}\frac{f(x)}{x} = 0, \qquad \lim_{x\to+\infty}\frac{f(x)}{x} = \infty.$$

Copyright © 2009 **А.Г. Гринь.**
Омский государственный университет.
E-mail: griniran@gmail.com

Если $f(x)$ – N-функция, то сопряженная к ней функция

$$f^*(y) = \max_{x>0}(xy - f(x))$$

также является N-функцией и $(f^*)^* = f$.

Пусть $(\Omega, \mathcal{F})$ – измеримое пространство, $\xi : \Omega \to \mathbb{R}$ – измеримая функция и $\mathbf{P}$– конечная мера на $\mathcal{F}$. Обозначим

$$\mathbf{E}_{\mathbf{P}}\xi = \mathbf{E}\xi = \int\limits_{\Omega} \xi(\omega)\, \mathbf{P}(d\omega)$$

(индекс $\mathbf{P}$ будем опускать там, где это не вызовет путаницы).

Пусть $f(x)$ – N-функция. Нормой Орлича функции $\xi(\omega)$ относительно меры $\mathbf{P}$ назовем число

$$\|\xi\|_f = \|\xi\|_{f,\mathbf{P}} = \sup\{\mathbf{E}_{\mathbf{P}}\xi\eta : \mathbf{E}_{\mathbf{P}} f^*(|\eta|) \le 1\},$$

а нормой Люксембурга –

$$\|\xi\|_{(f)} = \|\xi\|_{(f,\mathbf{P})} = \inf\left\{s > 0 : \mathbf{E}_{\mathbf{P}} f\left(\frac{|\xi|}{s}\right) \le 1\right\}.$$

Эти две нормы эквивалентны:

$$\|\xi\|_{(f)} \le \|\xi\|_f\| \le 2\xi\|_{(f)}. \tag{2}$$

Имеет место соотношение

$$\|\xi\|_{f,\mathbf{P}} \le \mathbf{E}_{\mathbf{P}} f(|\xi|) + 1. \tag{3}$$

Отметим формулу для нормы индикатора: если $\mathbf{1}_A(\omega) = \begin{cases} 1, & \omega \in A \\ 0, & \omega \notin A \end{cases}$, то

$$\|\mathbf{1}_A(\omega)\|_{(f,\mathbf{P})} = \frac{1}{f^{-1}\left(\dfrac{1}{\mathbf{P}(A)}\right)}. \tag{4}$$

Имеют место неравенство Гельдера и усиленное неравенство Гельдера:

$$|\mathbf{E}\xi\eta| \le \|\xi\|_f \|\eta\|_{f^*}, \tag{5}$$

$$|\mathbf{E}\xi\eta| \le \|\xi\|_{(f)} \|\eta\|_{f^*}. \tag{6}$$

Говорят, что N-функция $f(x)$ удовлетворяет Δ'-условию, если существуют постоянные $a > 0$ и $x_0 > 0$ такие, что

$$f(xy) \le a f(x) f(y), \quad x, y \ge x_0. \tag{7}$$

Теорема 1. *Пусть последовательность $\{\xi_n\}$ удовлетворяет условию φ-перемешивания, ξ и η измеримы соответственно относительно $\mathcal{F}_{\le 0}$ и $\mathcal{F}_{\ge n}$, а $f(x)$ – N-функция, удовлетворяющая Δ'-условию. Если $\mathbf{E}f(|\xi|) < \infty$ и $\mathbf{E}f^*(|\eta|) < \infty$, то*

$$|\mathbf{E}\xi\eta - \mathbf{E}\xi\,\mathbf{E}\eta| \le C\psi(n)\|\xi\|_f\|\eta\|_{f^*}, \quad \psi(n) = \frac{C}{f^{-1}\left(\dfrac{1}{\varphi(n)}\right)},$$

а константа C зависит только от f.

Замечание 1. Если $f(x) = p^{-1}x^p,\ p > 1$, то нетрудно подсчитать, что $f^*(x) = q^{-1}x^q,\ q > 1,\ p^{-1} + q^{-1} = 1,\ \|\xi\|_f = q^{1/q}\|\xi\|_p,\ \|\eta\|_{f^*} = p^{1/p}\|\eta\|_q$ (см., например, [2, с. 86]). Ясно, что $f(x)$ удовлетворяет Δ'-условию и $\psi(n) = Cp^{-1/p}\varphi^{1/p}(n)$. Мы видим, что в данном случае неравенство в теореме отличается от неравенства Ибрагимова лишь константой в правой части.

Доказательство. Докажем утверждение теоремы для величин $\xi \ge 0$ и $\eta \ge 0$ вида

$$\xi = \sum_{i=1}^{k} u_i \mathbf{1}_{A_i},\ A_i \in \mathcal{F}_{\le 0},\ i = 1, ..., k,\ A_iA_j = \varnothing, i \ne j,\ \bigcup_{i=1}^{k} A_i = \Omega,$$

$$\eta = \sum_{j=1}^{l} v_j \mathbf{1}_{B_j},\ B_j \in \mathcal{F}_{\ge n},\ j = 1, ..., l,\ B_iB_j = \varnothing, i \ne j,\ \bigcup_{j=1}^{l} B_j = \Omega. \tag{8}$$

С помощью неравенства Гельдера (5) получаем

$$|\mathbf{E}\xi\eta - \mathbf{E}\xi\,\mathbf{E}\eta| = |\mathbf{E}\xi(\mathbf{E}(\eta|\xi) - \mathbf{E}\eta)| \le \|\xi\|_f\|\mathbf{E}(\eta|\xi) - \mathbf{E}\eta\|_g, \tag{9}$$

где $g = f^*$, а $\mathbf{E}(\eta|\xi) = \sum\limits_{i=1}^{k}\left(\sum\limits_{j=1}^{l} v_j\mathbf{P}(B_j|A_i)\right)\mathbf{1}_{A_i}(\omega)$. При $\omega \in A_i$

$$|\mathbf{E}(\eta|\xi) - \mathbf{E}\eta| = \left|\sum_{j=1}^{l} v_j(\mathbf{P}(B_j|A_i) - \mathbf{P}(B_j))\right| = \mathbf{E}_{\mathbf{Q}_i^+}\eta + \mathbf{E}_{\mathbf{Q}_i^-}\eta, \tag{10}$$

где меры $\mathbf{Q}_i^+$ и $\mathbf{Q}_i^-$ определяются соотношениями

$$\mathbf{Q}_i^+(B_j) = (\mathbf{P}(B_j|A_i) - \mathbf{P}(B_j)) \vee 0,\ \mathbf{Q}_i^-(B_j) = (\mathbf{P}(B_j) - \mathbf{P}(B_j|A_i)) \vee 0,$$

$$a \vee b = \max(a, b), \quad \mathbf{Q}_i^{\pm}\left(\bigcup_j B_j\right) = \sum_j \mathbf{Q}_i^{\pm}(B_j).$$

Из усиленного неравенства Гельдера (6) следует

$$\mathbf{E}_{\mathbf{Q}_i^+}\eta + \mathbf{E}_{\mathbf{Q}_i^-}\eta \le \|1\|_{(f,\mathbf{Q}_i^+)}\|\eta\|_{g,\mathbf{Q}_i^+} + \|1\|_{(f,\mathbf{Q}_i^-)}\|\eta\|_{g,\mathbf{Q}_i^-}. \tag{11}$$

В силу (4)

$$\|1\|_{(f,\mathbf{Q}_i^+)} = \|\mathbf{1}_\Omega\|_{(f,\mathbf{Q}_i^+)} = \frac{1}{f^{-1}\left(\dfrac{1}{\mathbf{Q}_i^+(\Omega)}\right)},$$

$$\mathbf{Q}_i^+(\Omega) = \sum_{j=1}^{l} \mathbf{Q}_i^+(B_j) = \sum_j{}^+ (\mathbf{P}(B_j|A_i) - \mathbf{P}(B_j)) =$$

$$= \mathbf{P}\left(\bigcup_j{}^+ B_j|A_i\right) - \mathbf{P}\left(\bigcup_j{}^+ B_j\right) \le \varphi(n)$$

(здесь $\sum^+$ означает суммирование по положительным слагаемым, а в $\bigcup^+$) суммирование идет по тем же индексам, что и в $\sum^+$). Добавив к этому аналогичные рассуждения с $\mathbf{Q}_i^-$, получим

$$\|1\|_{(f,\mathbf{Q}_i^+)} \le \frac{1}{f^{-1}\left(\dfrac{1}{\varphi(n)}\right)}, \quad \|1\|_{(f,\mathbf{Q}_i^-)} \le \frac{1}{f^{-1}\left(\dfrac{1}{\varphi(n)}\right)}.$$

Вместе с (9), (10) и (11) это дает нам

$$|\mathbf{E}\xi\eta - \mathbf{E}\xi\,\mathbf{E}\eta| \le \frac{\|\xi\|_{f,\mathbf{P}}}{f^{-1}\left(\dfrac{1}{\varphi(n)}\right)} \left\| \sum_{i=1}^{k} \left(\|\eta\|_{g,\mathbf{Q}_i^+} + \|\eta\|_{g,\mathbf{Q}_i^-} \right) \mathbf{1}_{A_i} \right\|_{g,\mathbf{P}} \le$$

$$\le \frac{\|\xi\|_{f,\mathbf{P}}}{f^{-1}\left(\dfrac{1}{\varphi(n)}\right)} \left(S^+ + S^-\right), \quad S^\pm = \left\| \sum_{i=1}^{k} \|\eta\|_{g,\mathbf{Q}_i^\pm} \mathbf{1}_{A_i} \right\|_{g,\mathbf{P}}. \tag{12}$$

Оценим S^+, S^- оценивается аналогично. Согласно определению нормы Орлича, найдется функция $\zeta = \sum\limits_{i=1}^{k} y_i \mathbf{1}_{A_i}$, $y_i \ge 0$ такая, что

$$\mathbf{E}f(\zeta) = \sum_{i=1}^{k} f(y_i)\mathbf{P}(A_i) \le 1, \quad S^+ \le 2\sum_{i=1}^{k} y_i \|\eta\|_{g,\mathbf{Q}_i^+} \mathbf{P}(A_i), \tag{13}$$

и функции $\theta_i = \sum\limits_{j=1}^{l} x_{ij} \mathbf{1}_{B_j}$, $x_{ij} \ge 0$, $i = 1, ..., k$ такие, что

$$\mathbf{E}_{\mathbf{Q}_i^+} f(\theta_i) = \sum_{j=1}^{l} f(x_{ij}) \mathbf{Q}_i^+(B_j) \le 1, \quad \|\eta\|_{g,\mathbf{Q}_i^+} \le 2\sum_{j=1}^{l} v_j x_{ij} \mathbf{Q}_i^+(B_j). \tag{14}$$

Из (13) и (14) с помощью (6) получаем

$$S^+ \le 4\sum_{i=1}^{k}\sum_{j=l}^{l} x_{ij} y_i v_j \mathbf{Q}_i^+(B_j)\mathbf{P}(A_i) = 4\mathbf{E}_{\mathbf{R}}\eta\zeta\theta \le 4\|\eta\|_{(g,\mathbf{R})}\|\zeta\theta\|_{f,\mathbf{R}}. \tag{15}$$

Здесь $\theta = \sum\limits_{i=1}^{k}\sum\limits_{j=l}^{l} x_{ij}\mathbf{1}_{A_iB_j}$, а мера $\mathbf{R}$ определяется следующим образом: $\mathbf{R}(A_iB_j) =$

$$= \mathbf{Q}_i^+(B_j)\mathbf{P}(A_i) = (\mathbf{P}(A_iB_j) - \mathbf{P}(A_i)\mathbf{P}(B_j)) \vee 0, \quad \mathbf{R}\left(\bigcup_{i,j} A_iB_j\right) = \sum_{i,j}\mathbf{R}(A_iB_j).$$

Ясно, что $\mathbf{R}(A_iB_j) \le \mathbf{P}(A_iB_j)$, откуда, учитывая (2), выводим

$$\|\eta\|_{(g,\mathbf{R})} \le \|\eta\|_{(g,\mathbf{P})} \le \|\eta\|_{g,\mathbf{P}}. \tag{16}$$

Далее, из (3) следует

$$\|\zeta\theta\|_{f,\mathbf{R}} \le 1 + \mathbf{E}_{\mathbf{R}} f(\zeta\theta) = 1 + \sum_{i=1}^{k}\sum_{j=l}^{l} f(x_{ij}y_j)\mathbf{R}(A_iB_j) =$$

$$= 1 + \left(\sum\nolimits_1 + \sum\nolimits_2 + \sum\nolimits_3 + \sum\nolimits_4\right) f(x_{ij}y_j)\mathbf{R}(A_iB_j), \tag{17}$$

где в $\sum_i$, $i = 1,2,3,4$ суммирование производится соответственно по

$$\{i,j : x_{ij} \le x_0,\ y_i \le x_0\}, \quad \{i,j : x_{ij} \le x_0,\ y_i > x_0\},$$

$$\{i,j : x_{ij} > x_0,\ y_i \le x_0\}, \quad \{i,j : x_{ij} > x_0,\ y_i > x_0\},$$

а x_0 – константа из Δ'-условия (7). Учитывая (13) и (14), получаем

$$\sum\nolimits_1 f(x_{ij}y_j)\mathbf{R}(A_iB_j) \le f(x_0^2), \quad \sum\nolimits_2 f(x_{ij}y_j)\mathbf{R}(A_iB_j) \le$$

$$\le \sum_{i,j} f(x_0y_j)\mathbf{R}(A_iB_j) \le af(x_0)\sum_{i=1}^{k} f(y_i)\mathbf{P}(A_i) \le af(x_0),$$

$$\sum\nolimits_3 f(x_{ij}y_j)\mathbf{R}(A_iB_j) \le af(x_0)\sum_{i=1}^{k}\sum_{j=1}^{l} f(x_{ij})\mathbf{Q}_i^+(B_j)\mathbf{P}(A_i) \le af(x_0),$$

$$\sum\nolimits_4 f(x_{ij}y_j)\mathbf{R}(A_iB_j) \le a\sum_{i=1}^{k} f(y_i)\mathbf{P}(A_i)\sum_{j=1}^{l} f(x_{ij})\mathbf{Q}_i^+(B_j) \le a.$$

Таким образом, $\|\zeta\theta\|_{f,\mathbf{R}} \le C_1 = 1 + f(x_0) + a(1 + 2f(x_0))$, что вместе с (9), (12), (15) и (16) означает, что

$$|\mathbf{E}\xi\eta - \mathbf{E}\xi\,\mathbf{E}\eta| \le 8C_1\psi(n)\|\xi\|_f\|\eta\|_{f^*}, \tag{18}$$

где ξ и η функции вида (8).

Если $\xi \ge 0$, $\eta \ge 0$ измеримы соответственно относительно $\mathcal{F}_{\le 0}$ и $\mathcal{F}_{\ge n}$, то существуют последовательности функций $\{\xi_n\}$ и $\{\eta_n\}$ вида (8) и такие, что $\xi_n(\omega) \uparrow \xi(\omega)$, $\eta_n(\omega) \uparrow \eta(\omega)$, $n \to \infty$ при любом $\omega \in \Omega$. И если $\mathbf{E}f(|\xi|) < \infty$ и $\mathbf{E}f^*(|\eta|) < \infty$, то по теореме о мажорируемой сходимости $\mathbf{E}\xi_n \to \mathbf{E}\xi$,

$\mathbf{E}\eta_n \to \mathbf{E}\eta,\ \mathbf{E}\xi_n\eta_n \to \mathbf{E}\xi\eta,\ \|\xi_n\|_{f,\mathbf{P}}\| \to \xi\|_{f,\mathbf{P}},\ \|\eta_n\|_{f,\mathbf{P}} \to \|\eta\|_{f,\mathbf{P}},\ n \to \infty$, что вместе с доказанным выше означает, что (18) выполнено для данных $\xi \geq 0$ и $\eta \geq 0$.

Пусть, наконец, ξ и η – произвольные величины, удовлетворяющие условиям теоремы. Тогда $\xi = \xi^+ - \xi^-,\ \eta = \eta^+ - \eta^-,\ \xi^\pm = (\pm\xi) \vee 0,\ \eta^\pm = (\pm\eta) \vee 0$ и

$$|\mathbf{E}\xi\eta - \mathbf{E}\xi\,\mathbf{E}\eta| \leq |\mathbf{E}\xi^+\eta^+ - \mathbf{E}\xi^+\,\mathbf{E}\eta^+| + |\mathbf{E}\xi^+\eta^- - \mathbf{E}\xi^+\,\mathbf{E}\eta^-| +$$

$$+|\mathbf{E}\xi^-\eta^+ - \mathbf{E}\xi^-\,\mathbf{E}\eta^+| + |\mathbf{E}\xi^-\eta^- - \mathbf{E}\xi^-\,\mathbf{E}\eta^-|.$$

Применив к слагаемым в правой части последнего соотношения неравенство (18), получим утверждение теоремы с $C = 32C_1$. ■

Основное предназначение теоремы 1 – это использование техники, основанной на применении аналогов неравенств (1) в ситуациях, когда одна из величин ξ или η не имеет моментов порядка, большего 1 и, следовательно, нет возможности использовать соотношение (1). Однако если $\mathbf{E}|\xi| < \infty$, то можно построить N-функцию $f(x)$ такую, что $\mathbf{E}f(\xi) < \infty$ [2, с. 76]. Покажем один из возможных способов построения N-функции $f(x)$, удовлетворяющей Δ'-условию со сколь угодно медленным ростом $f(x)/x$.

Пусть функция $\varepsilon(t) > 0$ такова, что $\varepsilon(t) \downarrow 0,\ t \to +\infty$, и при некотором $c > 0$

$$\varepsilon(t) + \varepsilon^2(t) + t\varepsilon'(t) \geq 0,\ t \geq c, \quad \int\limits_c^\infty \frac{\varepsilon(t)}{t}\,dt = \infty. \tag{19}$$

Можно взять, например, в качестве $\varepsilon(t)$ медленно меняющуюся функцию, в этом случае $t\varepsilon'(t) = o(\varepsilon(t)),\ t \to \infty$ [3, с. 15] и, следовательно, выполняется первое соотношение в (19). При этом $\varepsilon(t)$ можно подобрать так, чтобы обеспечить сколь угодно медленный рост $\int\limits_c^x \frac{\varepsilon(t)}{t}\,dt$. Положим $f(x) = x\exp\left\{\int\limits_c^x \frac{\varepsilon(t)}{t}\,dt\right\}, x \geq c$. То, что $f(x)$ является N-функцией, проверяется без труда. Поясним лишь доказательство Δ'-условия. При $x, y \geq c \geq 1$ имеем

$$\int\limits_x^{cx} \frac{\varepsilon(t)}{t}\,dt \leq \varepsilon(x)\int\limits_x^{cx} \frac{dt}{t} = o(1),\ x \to \infty,$$

так что

$$\int\limits_c^{xy} \frac{\varepsilon(t)}{t}\,dt = \int\limits_c^x \frac{\varepsilon(t)}{t}\,dt + \int\limits_x^{cx} \frac{\varepsilon(t)}{t}\,dt + \int\limits_{cx}^{xy} \frac{\varepsilon(t)}{t}\,dt \leq \int\limits_c^x \frac{\varepsilon(t)}{t}\,dt + \int\limits_c^y \frac{\varepsilon(t)}{t}\,dt + o(1),$$

откуда следует Δ'-условие для функции $f(x)$.

Литература

1. Ибрагимов, И.А. Независимые и стационарно связанные величины / И.А. Ибрагимов, Ю.В. Линник. – М.: Наука, 1965.
2. Красносельский, М.А. Выпуклые функции и пространства Орлича / М.А. Красносельский, Я.Б Рутицкий. – М.: Изд-во физ.-мат. литературы, 1958.
3. Сенета, Е. Правильно меняющиеся функции / Е. Сенета. – М.: Наука, 1985.

Математические структуры и моделирование
2009, вып. 20, с. 12–26

УДК 513.813

ФОРМУЛЫ ТИПА ГАУССА-БОННЕ-ЧЕРНА ДЛЯ ПСЕВДОРИМАНОВЫХ И РИМАНОВЫХ МНОГООБРАЗИЙ И ФОРМУЛА ХИРЦЕБРУХА

А.К. Гуц

Дается обзор формул типа Гаусса-Бонне-Черна для римановых и псевдоримановых многообразий.

Классическая формула Гаусса-Бонне

$$\frac{1}{2\pi}\int\limits_F Kd\sigma = \chi(F) \tag{1}$$

связывает гауссову кривизну K замкнутой ориентированной двумерной поверхности F с характеристикой Эйлера-Пуанкаре поверхности F. Внутренняя геометрия, искривленность поверхности, увязаны в этой формуле с топологией поверхности, т.е. с ее формой.

Для односвязной области F с краем ∂F, состоящим из конечного числа гладких кривых с углами α_j в вершинах, формула Гаусса-Бонне имеет вид

$$\int\limits_F Kd\sigma + \int\limits_{\partial F} k_g ds = 2\pi + \sum_j (\pi - \alpha_j), \tag{2}$$

где k_g – геодезическая кривизна.

Гаусс [1] доказал её для геодезических треугольников, а Бонне [2] в более общем виде (2). В виде (1) формула Гаусса-Бонне была установлена фон Диком [3] в 1888 году [4].

Обобщённую формулу Гаусса-Бонне на случай римановых многообразий впервые установили Аллендорфер и А.Вейль [5] в 1943 году. Под влиянием Вейля [4] новое простое доказательство формулы дал в 1944 году Черн [6].

Для замкнутых псевдоримановых многообразий формулу независимо вывели Черн (1963, [7]) и Авец (1962, [9]).

К сожалению, несмотря на важность этой формулы для теории римановых многообразий, эта её форма плохо представлена в русско-язычной учебной литературе. А если и представлена, то в форме равенства когомологических классов, что является препятствием для её использования со стороны многочисленной группы читателей, не владеющих методами алгебраической топологии.

Copyright © 2009 **А.К. Гуц.**
Омский государственный университет.
E-mail: aguts@mail.ru

1. Псевдоримановы многообразия

1.1. Формула Гаусса-Бонне-Черна для замкнутых псевдоримановых многообразий M^{2k}

Пусть $< M^{2k}, g >$ – $2k$-мерное компактное ориентированное псевдориманово многообразие сигнатуры $< \underbrace{+...+}_{p}\underbrace{-...-}_{2k-p} >$ без края.

Пусть $dv = \sqrt{|det(g)|}dx^1 \wedge ... \wedge dx^{2k}$ – $2k$-форма объёма и

$$\Delta(R) = \frac{(-1)^{[p/2]}}{2^{2k}\pi^k k!}\eta^{i_1 i_2 ... i_{2k}}_{j_1 j_2 ... j_{2k}} Ri_1 i_2 j_1 j_2 Ri_3 i_4 j_3 j_4 ... Ri_{2k-1} i_{2k} j_{2k-1} j_{2k},$$

где

$$\eta^{i_1 i_2 ... i_{2k}}_{j_1 j_2 ... j_{2k}} = \begin{cases} +1, \{i_1 i_2 ... i_{2k}\} \text{ четная перестановка } \{j_1 j_2 ... j_{2k}\}, \\ -1, \{i_1 i_2 ... i_{2k}\} \text{ нечетная перестановка } \{j_1 j_2 ... j_{2k}\}, \\ 0, \text{ среди } \{i_1 i_2 ... i_{2k}\} \text{ или среди } \{j_1 j_2 ... j_{2k}\} \text{ есть одинаковые.} \end{cases}$$

Тогда [7, 9]

$$\boxed{\int\limits_{M^{2k}} \Delta(R)dv = \chi(M^{2k}),} \tag{1.1}$$

где $\chi(M^{2k})$ – характеристика Эйлера-Пуанкаре многообразия M^{2k}.

1.2. Формула Гаусса-Бонне-Черна для замкнутых псевдоримановых многообразий M^4

Пусть $< M^4, g >$ – 4-мерное замкнутое ориентированное псевдориманово многообразие сигнатуры $< \underbrace{+...+}_{p}\underbrace{-...-}_{4-p} >$.

Тогда [10]

$$\boxed{\int\limits_{M^4} W_{iklm}W^{iklm}dv = 2\int\limits_{M^4}\left(R^i_j R^j_i - \frac{1}{3}R^2\right)dv + (-1)^{p+[\frac{p}{2}]}8\pi^2\chi(M^4),} \tag{1.2}$$

где W – тензор Вейля.

1.3. Формула Черна-Гаусса-Бонне для псевдоримановых многообразий M^{2k} с краем

Пусть M^{2k} – псевдориманово многообразие с краем ∂M^{2k} и $q : T_1 M^{2k} \to M^{2k}$ – расслоение на сферы, ассоциированное с касательным расслоением TM^{2k} (т.е.

состоящее из векторов касательного расслоения с нормой 1). Существует $(2k-1)$-форма σ на T_1M^{2k}, для которой

$$\int\limits_{q^{-1}(x)} \sigma = 1 \quad \text{для всех } x \in M^{2k}$$

и $q^*Pf_k(\Omega) = d\sigma$.

Всякое векторное поле T, нормальное к ∂M^{2k}, задает несингулярное сечение $\tau : \partial M^{2k} \to T_1M^{2k}$.

Тогда имеет место формула Черна-Гаусса-Бонне для псевдоримановых многообразий M^{2k} с краем [11]:

$$\boxed{\int\limits_{M^{2k}} Pf(\Omega) = ind_{\partial M^{2k}}T + \int\limits_{\partial M^{2k}} \tau^*\sigma,} \tag{1.3}$$

где $ind_{\partial M^{2k}}$ определяется следующим образом.

Если T не нулевое векторное поле на ∂M^{2k}, $\overline{T}$ – продолжение векторного поля T на все многообразие M^{2k} и $a_1, ..., a_k$ – конечное число особых точек (нулей) поля $\overline{T}$ на M^{2k} [12, с. 516], то

$$ind_{\partial M^{2k}}T = \sum_{j=1}^{k} ind_{a_j}\overline{T}.$$

Если поле T трансверсально (в частности, нормально) к ∂M^{2k}, то

$$ind_{\partial M^{2k}}T = \chi(M^{2k}).$$

В общем случае для ориентированного компактного многообразия с краем:

$$ind_{\partial M^{2k}}T = \chi(M^{2k}) - deg(K_T),$$

где $deg(K_T)$ – степень отображения $K_T : \partial M^{2k} \to S^{2k-1}$ [12, с. 502], $K_T(x)$ равно точке на S^{2k-1}, отмеченной концом вектора $v = v^0T + v^1u_1 + ... + v^{2k-1}u_{2k-1}$, $\{u_1, ..., u_{2k-1}\}$ – базис в $T_x(\partial M^{2k})$, $x \in \partial M^{2k}$, $\sum_j v^j = 1$ [13].

Если Q^{2k} – $2k$-мерная компактная область в M^{2k} с границей ∂Q^{2k}, то формулу (1.3) можно переписать в виде

$$\boxed{\int\limits_{Q^{2k}} Pf(\Omega) = ind_{\partial Q^{2k}}T + \int\limits_{\partial Q^{2k}} \tau^*\sigma,} \tag{1.4}$$

где все формы, поля, определяются как выше с заменой буквы M на букву Q.

1.4. Формула для чисел Понтрягина для замкнутых псевдоримановых многообразий M^{4k}

Пусть $< M^{4k}, g >$ – $4k$-мерное замкнутое ориентированное псевдориманово многообразие сигнатуры $< \underbrace{+...+}_{p}\underbrace{-...-}_{4k-p} >$.

Пусть

$$dv = \sqrt{|det(g_{ij})|}dx^1 \wedge ... \wedge d^{4k}$$

– форма объёма и

$$\Delta_{4k} = \frac{(-1)^p}{(2\pi)^{2k}(2k)!}\Omega^{i_1}_{i_2} \wedge \Omega^{i_2}_{i_3} \wedge ... \wedge \Omega^{i_{2k}}_{i_1}$$

или

$$\Delta_{4k} = \frac{(-1)^p}{(2\pi)^{2k}(2k)!2^{2k}}R^{\alpha}_{\beta,\lambda_1\lambda_2}R^{\beta}_{\gamma,\lambda_3\lambda_4}...R^{\delta}_{\alpha,\lambda_{4k-1}\lambda_{4k}}\eta^{\lambda_1...\lambda_{4^k}}.$$

Тогда имеет место формула Занда [14] для k-мерного числа Понтрягина

$$\boxed{p_k[M^{4k}] = \int\limits_{M^{4k}} \Delta_{4k}\ dv.} \tag{1.5}$$

Напомним, что k-мерное число Понтрягина $p_k[M^{4k}]$ определяется как значение характеристического класса Понтрягина $p_k(\xi) \in H^{4k}(M^{4k}, \mathbb{R})$, где ξ – касательное расслоение на M^{4k} на фундаментальном классе $\mu_M \in H_{4k}(M^{4k}, \mathbb{R})$, т.е.

$$p_k[M^{4k}] = \langle p_k(\xi), \mu_M \rangle$$

или

$$p_k[M^{4k}] = \int\limits_{M^{4k}} p_k(\xi)$$

в случае, когда когомологический класс $p_k(\xi)$ реализуется как внешняя дифференциальная $4k$-форма, т.е. имеются в виду когомологии де Рама.

2. Римановы многообразия

Многообразие $< M^n, g >$ – риманово, если оно снабжено положительно определенной метрикой g.

2.1. Формула Гаусса-Бонне-Черна для замкнутых римановых многообразий M^{2k}

Пусть $< M^{2k}, g >$ – $2k$-мерное компактное ориентированное риманово многообразие без края и

$$\Omega = \frac{(-1)^k}{2^{2k}\pi^k k!}\eta^{i_1 i_2 ... i_{2k}}_{12...(2k)}\Omega i_1 i_2 \wedge ... \wedge \Omega i_{2k-1} i_{2k}.$$

Имеем

$$\Omega_{ij} = \frac{1}{2} R_{ijkl} dx^k \wedge dx^l$$

и

$$\Omega = E_{2k} dv,$$

где $dv = \sqrt{g} dx^1 \wedge ... \wedge dx^{2k}$ – форма объёма и (см. [16, p. 105]):

$$E_{2k} = \frac{(-1)^k}{2^{2k}(2\pi)^k k!} \eta^{i_1 i_2 ... i_{2k}}_{j_1 j_2 ... j_{2k}} R i_1 i_2 j_1 j_2 ... R i_{2k-1} i_{2k} j_{2k-1} j_{2k}.$$

Здесь (см. [20, с. 56])

$$\eta^{i_1 i_2 ... i_{2k}}_{j_1 j_2 ... j_{2k}} = \begin{cases} +1, & \{i_1 i_2 ... i_{2k}\} \text{ различны и дают четную перестановку } \{j_1 j_2 ... j_{2k}\}, \\ -1, & \{i_1 i_2 ... i_{2k}\} \text{ различны и дают нечетную перестановку } \{j_1 j_2 ... j_{2k}\}, \\ 0, & \text{среди } \{i_1 i_2 ... i_{2k}\} \text{ или среди } \{j_1 j_2 ... j_{2k}\} \text{ есть одинаковые.} \end{cases}$$

Тогда [16, p. 106], [20, p. 171]:

$$\boxed{\int\limits_{M^{2k}} E_{2k} dv = \chi(M^{2k}).} \qquad (2.1)$$

2.2. Формулы Ревентоса для замкнутого риманова многообразия M^{2n+1}

В 1972 году Танно [17] доказал следующую теорему.

Теорема 1. *Пусть $< M^{2n+1}, g, \xi >$ – компактное риманово ориентированное многообразие, имеющее единичное киллингово векторное поле ξ такое, что $R(X,\xi)Y = g(X,Y)\xi - g(Y,\xi)X$ для любых векторных полей X, Y на M^{2n+1}, т.е. является сасакиевым многообразием. Тогда, если поле ξ – регулярное* [19], *то*

$$\frac{(-1)^n}{l(\xi) 2^{2n} \pi^n n!} \int\limits_{M^{2n+1}} F(\Omega, \xi) = \sum_{r=0}^{n} (n+1-r)(-1)^r \beta_r(M^{2n+1}),$$

где $F(\Omega, \xi)$ – выражение, зависящее от Ω и ξ и $\beta_r(M^{2n+1})$ – r-мерное число Бетти, $l(\xi)$ – длина интегральной траектории поля ξ (она постоянна).

Теорема Танно была обобщена в 1979 году Ревентосом на произвольные римановы многообразия с симметрией.

Пусть $< M^{2n+1}, g, \xi >$ – замкнутое ориентированное риманово многообразие, имеющее единичное киллингово векторное поле ξ.

Тогда справедлива формула [18] в предположении, что поле ξ – регулярное[1] [19]:

$$\boxed{\frac{(-1)^n}{l(\xi)2^{2n}\pi^n n!}\int\limits_{M^{2n+1}} f(\Omega,\xi)=\sum_{r=0}^{n}(n+1-r)(-1)^r\beta_r(M^{2n+1})+\sum_{r=0}^{n-1}d_r,} \qquad (2.2)$$

где $d_r=\dim Ker(H^r(M^{2n+1}/\xi,\mathbb{R}),\mathbb{R})\to H^{r+2}(M^{2n+1}/\xi,\mathbb{R})$, $l(\xi)$ – длина интегральной траектории поля ξ (она постоянна).

Все интегральные кривые поля ξ гомеоморфны S^1, и можно построить главное расслоение

$$\pi: M^{2n+1}\to M^{2n+1}/\xi=B,$$

где B – множество всех орбит [19].

Выражение

$$f(\Omega,\xi)=\sum\eta^{i_1\dots i_{2n}}\left(\Psi^{i_2}_{i_1}+\sum_{ks}(A_{i_2i_1}A_{ks}+A_{i_2k}A_{i_1s})\phi^k\wedge\phi^s\right)\wedge\dots$$

$$\dots\wedge\left(\Psi^{i_{2n}}_{i_{2n-1}}+\sum_{ks}(A_{i_{2n}i_{2n-1}}A_{ks}+A_{i_{2n}k}A_{i_{2n-1}s})\phi^k\wedge\phi^s\right)\wedge\phi^0,$$

Ψ^i_j – форма кривизны на M^{2n+1} относительно метрики g,

$$\phi^0=\omega,\quad \phi^i=\pi^*(\theta^i),\quad g=\sum_\alpha\phi^\alpha\otimes\phi^\alpha,$$

ω – 1-форма, определенная на M^{2n+1} как $\omega(X)=g(\xi,X)$, $\theta^1,\dots,\theta^{2n}$ – 1-формы, заданные в малом открытом множестве $U\subset B$ так, что $h=\sum_i\theta^i\otimes\theta^i$, а h риманова метрика на B, определенная равенством

$$h(X,Y)(b)=g(X',Y')(x)-(\omega\otimes\omega)(X',Y')(x),$$

$$d\pi_x(X')(x)=X,\quad d\pi_x(Y')(x)=Y,\quad \pi(x)=b,\quad x\in M^{2n+1},$$

$$d\omega=\pi^*(\sum_{ij}A_{ij}\theta^i\wedge\theta^j)\ \text{ с }\ A_{ij}=-A_{ji}.$$

При этом $A_{ij}=-g(e_i,\nabla_{e_j}\xi)$, $\{e_1,\dots,e_{2n}\}$ – базис в $T(B)$.

Формула (2.2) получена Ревентосом [18] из формулы Гаусса-Бонне-Черна для замкнутого ориентированного риманова $2n$-мерного многообразия $<B,h>$:

$$\int\limits_B Q=2^{2n}\pi^n n!\chi(B),$$

[1] Условие регулярности выполняется, как правило, для всех векторных полей, встречающихся в приложениях.

где

$$Q = (-1)^n \sum \eta^{i_1 \dots i_{2n}} \Omega^{i_2}_{i_1} \wedge \dots \Omega^{i_{2n}}_{i_{2n-1}},$$

Ω – форма кривизны многообразия $< B, h >$, с помощью поднятия π^* и выражения $\chi(B)$ через числа Бетти $\beta(M^{2n+1})$.

2.2.1. Формула Ревентоса для замкнутого риманова многообразия M^3

В случая замкнутого ориентированного риманова мнгообразия M^3 с метрикой g, допускающего регулярное единичное киллингово векторное поле ξ, имеет место формула [18]:

$$\boxed{\frac{1}{2\pi l(\xi)} \int\limits_{M^3} \{K(\xi^{\perp}) + 3K(\xi)\} dv = 2\beta_0(M^3) - \beta_1(M^3) + d_0,} \tag{2.3}$$

где $d_0 = 0$ или 1 в зависимости от чётности или нечётности одномерного числа Бетти $\beta_1(M^3)$; $K(\xi^{\perp})$ – значение римановой кривизны в плоскости, ортогональной ξ; $K(\xi)$ – значение римановой кривизны для любой плоскости, содержащей ξ (отметим, что $K(\xi)$ не зависит от выбора плоскости); dv – форма объёма; $l(\xi)$ – длина интегральной траектории поля ξ (она постоянна).

2.3. Формула Черна-Гаусса-Бонне для римановых многообразий M^{2k} с краем

Пусть M^{2k} – компактное ориентированное риманово многообразие с краем. Пусть $\pi : TM^{2k} \to M^{2k}$ – касательное расслоение и $\pi_0 = \pi|S \to M^{2k}$ – ассоциированное расслоение на сферы (т.е. состоящее из векторов касательного расслоения с нормой 1). Пусть Γ – связность на главном расслоении $p : SO(TM^{2k}) \to M^{2k}$ с формой кривизны K и пусть Ω единственная $2k$-форма на M^{2k} такая, что $p^*\Omega = \pi^{-k} Pf(K)$ [2] и Φ – $(2k-1)$-форма на S с $\pi_0^*\Omega = d\Phi$. Наконец, пусть $\nu : \partial M^{2k} \to S$ – внешняя единичная нормаль к ∂M^{2k}.

Тогда [21]

$$\boxed{\int\limits_{M^{2k}} \Omega = \chi(M^{2k}) + \int\limits_{\partial M^{2k}} \nu^*\Phi.} \tag{2.4}$$

[2] Определение пфаффиана:

$$Pf(A) = \frac{1}{2^k k!} \varepsilon^{i_1 \dots i_{2k}} A_{i_1 i_2} A_{i_3 i_4} \dots A_{i_{2k-1} i_{2k}},$$

где $A = A_{ij}$.

2.4. Формула Черна-Гаусса-Бонне для римановых многообразий M^m с краем

Пусть M^m – ориентированное риманово многообразие с краем и

$$E_m = \frac{(-1)^p}{2^m \pi^p p!} \eta^{i_1 i_2 \ldots i_m}_{12\ldots(m)} \Omega_{i_1 i_2} \wedge \ldots \wedge \Omega_{i_{m-1} i_m},$$

$$p = \left[\frac{m}{2}\right],$$

$$Q_{k,m} = c_{k,m} \sum \eta^{i_1 i_2 \ldots i_{m-1}} \Omega_{i_1 i_2} \wedge \ldots \wedge \Omega_{i_{2k-1} i_{2k}} \wedge \omega_{i_{2k+1} m} \wedge \ldots \wedge \omega_{i_{m-1} m},$$

$$c_{k,m} = \frac{(-1)^k}{\pi^p k! 2^{k+p} \cdot 1 \cdot 3 \cdot \ldots \cdot (2p-2k-1)}.$$

Здесь

$$\Omega_{jk} = d\omega_{jk} - \sum_{1 \le \nu \le m} \omega_{j\nu} \wedge \omega_{\nu k},$$

$$\nabla e_j = \sum_{1 \le k \le m} \omega_{jk} e_k \quad \text{для} \quad \omega_{jk} \in T^*(M) \quad \text{и} \quad \omega_{jk} = -\omega_{kj},$$

$\{e_1, \ldots, e_m\}$ – локальный ортонормированный базис в $T(M)$ такой, что e_m – единичная нормаль к краю ∂M^m.

Справедлива формула Гаусса-Бонне-Черна [16, p. 253]:

$$\boxed{\int\limits_{M^m} E_m + \int\limits_{\partial M^m} \sum_k Q_{k,m} = \chi(M^m).} \tag{2.5}$$

Замечание 1. Напомним, что $\chi(M^{2k+1}) = 0$ и, как доказал Черн,

$$E_m = -d\left(\sum_k Q_{k,m}\right).$$

Следовательно, в силу теоремы Стокса равенство (2.5) обращается для нечётномерных многообразий в тождество $0 = 0$.

2.5. Формула Гаусса-Бонне-Черна для компактных римановых многообразий M^4 с краем

Имеет место следующая формула [22] Гаусса-Бонне-Черна:

$$\boxed{\begin{aligned} 8\pi^2\chi(M^4,\partial M^4) = \int\limits_{M^4}\left(|W^{(4)}|^2-\frac{1}{2}|Z^{(4)}|^2-\frac{1}{24}[R^{(4)}]^2\right)d^{(4)}v- \\ -4\int\limits_{\partial M^4}\prod_{i=1}^{3}\lambda_i d^3v-\int\limits_{\partial M^4}\sum_{\sigma_i\in S^3}K_{\sigma_1\sigma_2}\lambda_{\sigma_3}d^3v, \end{aligned}} \tag{2.6}$$

где $\chi(M^4,\partial M^4)$ – характеристика Эйлера-Пуанкаре многообразия с краем M^4, W – тензор Вейля, $Z = Ric - (R/4)g$, Ric – тензор Риччи, R – скалярная кривизна, $K_{\sigma_1\sigma_2}$ – секционная кривизна, λ_i – главные кривизны многообразия ∂M^4.

3. Сигнатурные формулы

3.1. Сигнатура многообразия

Сигнатура многообразия M^n определяется для $n = 4k$. Если M^{4k} связное и ориентированное, то $\cup$-произведение когомологий $H^{2k}(M^{4k},\mathbb{R})$:

$$H^{2k}(M^{4k},\mathbb{R})\cup H^{2k}(M^{4k},\mathbb{R})\to H^{4k}(M^{4k},\mathbb{R})$$

является коммутативным.

Если предположить, что M^{4k} компактно, то в силу двойственности Пуанкаре $H^{4k}(M^{4k}) \cong H^0(M^{4k})$. Поскольку $H^0(V^{4k})$ является одномерным векторным пространством, то считаем, что $H^0(M^{4k}) = \mathbb{R}$ и имеем следующую билинейную форму:

$$Q : H^{2k}(M^{4k},\mathbb{R})\cup H^{2k}(M^{4k},\mathbb{R})\to\mathbb{R}.$$

Таким образом, определена квадратичная форма $Q(\alpha,\alpha)$ со значениями в $\mathbb{R}$, где $\alpha\in H^{2k}(M^{4k},\mathbb{R})$. Ее сигнатура, т.е. разность $\sigma(M^{4k}) = p(M^{4k}) - n(M^{4k})$ между числом положительных $p(M^{4k})$ и отрицательных $n(M^{4k})$ собственных значений, называется *сигнатурой многообразия* M^{4k}.

В случае когомологий де Рама $\alpha,\beta\in H^{2k}_{dR}(M^{4k},\mathbb{R})$

$$Q(\alpha,\beta)=\int\limits_{M^{4k}}\alpha\wedge\beta.$$

3.2. Формулы для сигнатуры замкнутого псевдориманова многообразия

Если $\sigma(M^{4k}) = p(M^{4k}) - n(M^{4k})$ – сигнатура[3] псевдориманова многообразия $\langle M^{4k}, g\rangle$, то имеет место формула [14]:

$$\boxed{\sigma(M^{4k}) = \frac{(-1)^p}{(2\pi)^{2k}(2k)!3^k} \int\limits_{M^{4k}} \Delta_{4k}\ dv,} \tag{3.1}$$

где

$$\Delta_{4k} = \frac{(-1)^p}{(2\pi)^{2k}(2k)!2^{2k}} R^{\alpha}_{\beta,\lambda_1\lambda_2} R^{\beta}_{\gamma,\lambda_3\lambda_4} ... R^{\delta}_{\alpha,\lambda_{4k-1}\lambda_{4k}} \eta^{\lambda_1 ... \lambda_{4^k}}.$$

Для 4-мерного многообразия имеем:

$$\boxed{\sigma(M^4) = \frac{(-1)^p}{96\pi^2} \int\limits_{M^4} R^{i}_{\cdot jj_1j_2} R_{i\cdot j_3j_4}^{\ \ j} \eta^{j_1j_2j_3j_4} \cdot dv} \tag{3.2}$$

или

$$\boxed{\sigma(M^4) = \frac{(-1)^p}{96\pi^2} \int\limits_{M^4} W_{ijkl} W^{ij}_{\ \ mn} \eta^{klmn} \cdot dv.} \tag{3.3}$$

Замечание 2. Если M^4 – пространство-время, т.е. $p = 1$, а W принадлежит типу III Петрова, то, как известно (L. Bel, 1960, см. [10]), $W_{ijkl} W^{ij}_{\ \ mn} \eta^{klmn} = 0$. Значит, мы имеем $\sigma(M^4) = 0$, тогда как (метрическая) сигнатура $\tau(M^4) = -2$.

3.3. Формула Хирцебруха для сигнатуры многообразия

3.3.1. Теорема Хирцебруха и L-род

Существуют однозначно определенные полиномы $L = \{L_k\}$ такие, что для любого ориентированного многообразия выполняется *сигнатурная теорема Хир-*

[3]Под сигнатурой замкнутого псевдориманова многообразия часто понимают число $\tau(M^4) = p - (2k - p)$, где p – число, входящее в сигнатуру $< \underbrace{+...+}_{p}\underbrace{-...-}_{4k-p} >$. Числа $\sigma(M^{4k})$ и $\tau(M^4)$ – это разные числа и по смыслу, и по значениям, как видно из замечания 2.

цебруха:

$$\boxed{\sigma(M^{4k}) = L_k(p_1, p_2, ..., p_k)[M^{4k}]} \tag{3.4}$$

или

$$\boxed{\sigma(M^{4k}) = \sum_{\substack{(j_1,...j_k)\\ j_1+...+j_r=k}} l_k^{(j)} p_{j_1}...p_{j_r}[M^{4k}],} \tag{3.5}$$

где $p_j(\xi) \in H^{4j}(M^{4k}, \mathbb{R})$ – j-й класс Понтрягина,

$$L_k(p_1, p_2, ..., p_k) = \sum_{\substack{(j_1,...j_k)\\ j_1+...+j_r=k}} l_k^{(j)} p_{j_1}...p_{j_r}.$$

Обычно о содержании формул (3.4), (3.5) говорят, что *сигнатура многообразия равна L-роду.*

Первые четыре полинома имеют вид:

$$L_0 = 1, L_1 = p_1/3, L_2 = (7p_2 - p_1^2)/45,$$

$$L_3 = (62p_3 - 13p_2p_1 + 2p_1^3)/945.$$

Следовательно, сигнатура 4-мерного замкнутого многообразия равна

$$\sigma(M^4) = \frac{1}{45}(7p_2[M^4] - p_1^2[M^4]).$$

Имеется алгоритм [15, § 19] вычисления полиномов L_k, использующий ряд

$$\frac{\sqrt{z}}{\operatorname{th}\sqrt{z}} = \sum_{k\geq 0} \frac{2^{2k}B_{2k}z^k}{(2k)!} = 1 + z/3 - z^2/45 + ...$$

где B_{2k} – числа Бернулли.

В случае представления классов p_k в виде дифференциальных $4k$-форм, т.е. при переходе к когомологиям де Рама, $p_k \in H^{4k}_{dR}(M^{4k}, \mathbb{R})$, – формула (3.5) принимает вид

$$\boxed{\sigma(M^{4k}) = \sum_{\substack{(j_1,...j_k)\\ j_1+...+j_r=k}} l_k^{(j)} \int\limits_{M^{4k}} p_{j_1} \wedge ... \wedge p_{j_r}.} \tag{3.6}$$

3.3.2. Выражение классов Понтрягина через тензор кривизны

На римановом многообразии M^{4k} классы Понтрягина

$$p(\xi) = 1 + p_1(\xi) + p_2(\xi) + ... + p_k(\xi)$$

выражаются через кривизны [25]:

$$p_s(\xi) = \frac{1}{(2\pi)^{2s}(2s)!} \sum \eta^{j_1...j_{2s}}_{i_1...i_{2s}} \Omega^{i_1}_{j_1} \wedge ... \wedge \Omega^{i_{2s}}_{j_{2s}}. \tag{3.7}$$

Если использовать формулу

$$\Omega_{ij} = \frac{1}{2} R_{ijkl} dx^k \wedge dx^l$$

и подставить ее в формулу (3.7), то получаем представление классов Понтрягина в когомологиях де Рама в терминах тензора кривизны:

$$p_s(\xi) = \frac{1}{(4\pi)^{2s}(2s)!} \sum \eta^{j_1...j_{2s}}_{i_1...i_{2s}} R^{i_1}_{\cdot j_1 l_1 m_1} ... R^{i_{2s}}_{\cdot j_{2s} l_{2s} m_{2s}} dx^{l_1} \wedge dx^{m_1} \wedge ... \wedge dx^{l_{2s}} \wedge dx^{m_{2s}}.$$

3.3.3. Выражение чисел Понтрягина через тензор кривизны

Пусть дано замкнутое ориентированное многообразие M^{4k} и пусть $(j) = (j_1, ..., j_k)$ – разбиение числа k, т.е. все $j_i > 0$ и $j_1 + ... + j_r = k$. Определим число Понтрягина $p_{(j)}[M^{4k}]$ как число равное

$$\langle p_{j_1}(\xi) ... p_{j_1}(\xi), \mu_{M^{4k}} \rangle.$$

При представлении классов $p_k(\xi)$ в виде дифференциальных $4k$-форм, т.е. при переходе к когомологиям де Рама, это определение означает [25], что

$$\boxed{p_{(j)}[M^{4k}] = \int\limits_{M^{4k}} p_{j_r} \wedge ... \wedge p_{j_r} = \int\limits_{M^{4k}} g \cdot dv,} \tag{3.8}$$

где dv – форма объема и функция

$$g(x) = \frac{1}{c} \sum_{l_1,...,l_{4k}} \eta^{l_1...l_{4k}} F_{l_1...l_{4k}},$$

$$c = 2^{4k} \pi^{2k} \prod_{i=1}^{r} (2j_i)!(4j_i)!,$$

а $F_{l_1...l_{4k}}$ – произведение следующих r функций:

$$\sum \eta^{m_1\ldots m_{2j_1}}_{i_1\ldots i_{2j_1}} \sum\ sgn(\sigma) R_{l_{\sigma(1)}l_{\sigma(2)}i_1m_1}\ldots R_{l_{\sigma(4j_1-1)}l_{\sigma(4j_1)}i_{2j_1}m_{2j_1}},$$

.....

.....

$$\sum \eta^{m_1\ldots m_{2j_r}}_{i_1\ldots i_{2j_r}} \sum\ sgn(\sigma) R_{l_{\sigma(4k-j_r+1)}l_{\sigma(4k-j_r+2)}i_1m_1}\ldots R_{l_{\sigma(4k-1)}l_{\sigma(4k)}i_{2j_r}m_{2j_r}},$$

относительно любого ортонормированного базиса в касательном пространстве $T_x(M^{4k})$. Здесь σ обозначает перестановку в множествах индексов $\{1,...,4j_1\},\{4j_1+1,...,4(j_1+j_2)\},\{4k-j_r+1,...,4k\}$, определенных разбиением (j).

3.3.4. Формула Хирцебруха

Поскольку числа Понтрягина на замкнутом ориентированном римановом многообразии выражаются через кривизны Ω или R, то сигнатура многообразия вычисляется через кривизну, и имеет место *формула Хирцебруха*:

$$\boxed{\sigma(M^{4k}) = \int\limits_{M^{4k}} L_k\left(\frac{\Omega}{\pi}\right) dv} \tag{3.9}$$

или

$$\boxed{\sigma(M^{4k}) = \int\limits_{M^{4k}} L_k\left(\frac{R}{\pi}\right) dv.} \tag{3.10}$$

Литература

1. Gauss, C.F. Disquisitiones generales circa superficies curvas, 1827 / C.F. Gauss // In: Dombrowski P. '150 Years After Gauss' 'Disquisitiones generales circa superficies curvas', Astérisque 62. – Paris: Soc Math de France, 1979.
2. Bonnet, O. Mémoire sur la théorie générale des surfaces / O. Bonnet // J. de l'Ecole Polytechnique. 1848. V. 19(32). P. 1–146.
3. von Dyck, W. Beiträge zur analysis situs / W. von Dyck // Math. Ann. – 1888. – B. 32. – S. 457–512.
4. Wu, H. Historical development of the Gauss-Bonnet theorem / H. Wu // Science in China Series A: Mathematics Apr. – 2008. – V. 51, N. 4. – P. 777–784.
5. Allendoerfer, C.B. The Gauss-Bonnet theorem for Riemannian polyhedra / C.B. Allendoerfer, A. Weil // Trans. Amer. Math. Soc. – 1943. – V. 53. – P. 101–129.
6. Chern, S.S. A simple intrinsic proof of the Gauss-Bonnet formula for closed Riemannian manifolds / S.S. Chern // Ann. of Math. – 1944. – V. 45. – P. 747–752.

7. Chern, S.S. Pseudo-Riemannian Geometry and the Gauss-Bonnet Formula / S.S. Chern // Ann. Acad. Brasil Ci. – 1963. – V. 35. – P. 17-26.
8. Chern, S.S. Historical remarks on Gauss-Bonnet / S.S. Chern // In: Analysis, et cetera, Volume in Honor of Jourgen Moser. New York: Academic Press, 1990. – P. 209–217.
9. Avez, A. Formula de Gauss-Bonnet-Chern en métrique de signature quelconque / A. Avez // C.R. Acad. Sci. Paris. – 1962. – T. 255. – P. 2049-2051.
10. Avez, A. Characteristic Classes and Weyl Tensor: Applications to General Relativity / A. Avez // Proceedings of the National Academy of Sciences. – 1970. – V. 6, N. 2. – P. 265-268.
11. Pelletier, F. Quelques proprietes géométriques des variétés pseudo-riemanniennes singuliéres / F. Pelletier // Annales de la faculé des sciences de Toulouse 6^e série. – 1995. – T. 4, N. 1. – P. 87-199.
12. Дубровин, Б.А. Современная геометрия / Б.А. Дубровин, С.П. Новиков, Ф.Т. Фоменко. – М.:Наука, 1979.
13. Alty, L.J. The Generalised Gauss-Bonnet-Chern Theorem / L.J. Alty // J.Math.Phys. – 1995. – V. 36. – P. 3094-3105.
14. Zund, J.D. Pontjagin numbers and pseudo-Riemannian geometry / J.D Zund // Annali di Matematica Pura ed Applicata. – 1966. – V. 72, N. 1.– P. 319-324.
15. Милнор Дж., Сташеф Дж. Характеристические классы / Дж. Милнор, Дж. Сташеф. – М.: Мир, 1979.
16. Gilkey, P.B. Invariance theory, the heat equation, and the Atiyan-Singer index theorem [Электронный ресурс]. – Режим доступа: – http://www.emis.de/monographs/gilkey/ (13.10.09).
17. Tanno, S. A formula on some odd-dimensional Riemannian manifolds related to the Gauss-Bonnet formula / S. Tanno // J. Math. Soc. Japan. – 1972. – V. 24. – P. 204-212.
18. Revenós, A. On the Gauss-Bonnet formula on the odd-dimensional manifolds / A. Reventós // Tohoku Math. J. – 1979. – V. 31, N. 2. – P. 165-178.
19. Palais, R. A global formulation of the Lie theory of transformation groups / R. Palais // Memoir of the Amer. Math. Soc. – 1957. – N. 22.
20. Chern, S.S. Lectures on Differential geometry / S.S. Chern, W.H. Chen, K.S. Lam. – World Scientific, 2000.
21. Spivak, M. A Comprehensive Introduction to Differential Geometry. V.5 / M. Spivak – Berkeley: Publish or Perish inc., 1979.
22. Anderson, M.T. L^2 Curvature and Volume renormalization of AHE metrics on 4-manifolds / M.T. Anderson // Mathematical Research Letters. – 2001. – V. 8. – P. 171–188.
23. Bao, D. A Note on the Gauss-Bonnet theorem for Finsler spaces / D. Bao, S.S. Chern // Ann of Math. – 1996. – V. 143(2). – P. 233–252.
24. Bell, D. The Gauss-Bonnet theorem for vector bundles [Электронный ресурс]. – Режим доступа: arXiv:math/0702162v1 (2007).
25. Galaz-Garcia, E. Bounds of characteristic numbers by curvarure and radius / E. Galaz-Garsia // Rocky Mountain J. Math. – 2009. – V. 39, N. 4. – P. 1225-1231.
26. Hsiung, C.-C. Curvature and characteristic classes of compact pseudo-Reimannian manifold / Chuan-Chih Hsiung, J.J. Levko // Rocky Mountain J. Math. – 1971. – V. 1, N. 3. – P. 523-536.

Математические структуры и моделирование
2009, вып. 20, с. 26–47

УДК 514.82

ОПИСАНИЕ ОДНОРОДНЫХ АФФИННЫХ ПРИЧИННЫХ ПОРЯДКОВ НА ТРЕХМЕРНЫХ РАЗРЕШИМЫХ ГРУППАХ ЛИ

Е.В. Мякишева

В статье исследуются однородные аффинные причинные порядки на трехмерных разрешимых группах Ли относительно аффинной структуры С.П. Гаврилова.

Введение

В данной работе ставилась задача исследования однородных аффинных причинных порядковых структур, задаваемых эллиптическими конусами на трехмерных разрешимых группах Ли, снабженных полной левоинвариантной аффинной канонической [5] структурой.

Однородные конусы в n-мерном аффинном пространстве изучались Э.Б. Винбергом [4]. Он алгебраически описал выпуклые конусы с острой вершиной, внутри которых транзитивно действует группа порядковых автоморфизмов $Aut(\mathcal{P})$. Семейство равных и параллельных конусов $\{C_x\}$ в A^n, $n \geq 3$, где C_x - множество лучей, исходящих из одной точки x, исследовал А.Д. Александров [3]. Он описал конусы C_x, на которых транзитивно действует группа Γ – биекций $f\colon A^n \to A^n$ таких, что $f(C_x) = C_{f(x)}$, удовлетворяющая условию: для любых $y \in C_x$, $y' \in C_{x'}$ существует $h \in \Gamma$ такая, что $h(x) = x'$ и $h(y) = y'$.

В настоящей работе рассматривались эллиптические конусы, задающие левоинвариантный аффинный причинный порядок относительно канонической аффинной структуры на трехмерных связных односвязных группах Ли. Проводилось исследование для выявления однородных порядков.

Исследование показало, что на группах Ли G_3I, G_3VI_0, G_3VII_0 класса 1 (в обозначениях С.П. Гаврилова [5]) аффинный причинный порядок является одновременно int - однородным, ∂ - однородным и ext - однородным. В остальных случаях порядок не является однородным ни в одном из указанных смыслов (Теорема 2).

Получен был также следующий результат. Если через H обозначим группу Ли аффинных преобразований относительно канонической аффинной структуры, сохраняющих изотропные векторы левоинвариантной лоренцевой метрики

Copyright © 2009 **Е.В. Мякишева.**
Омский государственный университет.
E-mail: elena-myakisheva@yandex.ru

g на G_3, то в случае групп Ли $G_3II - G_3VII$, класса 1 $H \subset Isom(G_3)$, для групп Ли G_3I, G_3VI_0, G_3VII_0 класса 1, когда метрика g плоская $H \subset Hom(G_3)$. В случае групп Ли $G_3II - G_3V$, G_3VII класса 2 $H \subset Hom(G_3)$. Для группы Ли G_3IV, класса 2 $H \subset Isom(G_3)$. Для группы Ли G_3VI, $H \subset Hom(G_3)$, за исключением того случая, когда метрика g является конформно-плоской (Теорема 3).

1. Аффинная структура групп Ли

Пусть V^n обозначает n - мерное дифференцируемое многообразие, $n \geqslant 1$, $\mathbb{R}^n$ - n - мерное арифметическое пространство.

Определение 1. Аффинный атлас $\mathcal{A}$ на V^n есть совокупность накрывающих V^n локальных карт таких, что каждая функция перехода между картами из $\mathcal{A}$ может быть продолжена до преобразования пространства $\mathbb{R}^n$. Максимальный аффинный атлас есть аффинная структура.

Определение 2. Аффинным многообразием называется V^n, оснащенное аффинной структурой.

Каждая локальная карта аффинной структуры определяет аффинные координаты.

Определение 3. Отображение $f\colon V^n \to V^m$, где V^n, V^m - аффинные многообразия, называется аффинным, если в локальных координатах аффинного атласа задается аффинным преобразованием из $\mathbb{R}^n$ в $\mathbb{R}^m$.

Множество всех аффинных отображений аффинного многообразия V^n в себя обозначим через $\mathrm{Aff}(V^n)$.

На аффинном многообразии V^n имеется естественная линейная связность ∇ с нулевыми кривизной и кручением, в аффинных координатах она является стандартной связностью на $\mathbb{R}^n$.
Аффинное многообразие можно определять через связность следующим образом.

Определение 4. Многообразие V^n допускает аффинную структуру, если на расслоение реперов многообразия V^n можно ввести линейную связность ∇ с тождественно равными нулю кривизной и кручением.

Если связность ∇ полная, то будем говорить о полной аффинной структуре. Геодезические связности ∇ назовем прямыми, полугеодезические - лучами.

Пусть V^n аффинное многообразие с лоренцевой метрикой g. Нас будут интересовать лоренцевы многообразия V^n с просто транзитивной разрешимой группой аффинных изометрий T. Следовательно, оснащение однородного V^n (полной) аффинной структурой равносильно оснащению (полной) левоинвариантной аффинной структурой группы Ли T. Левоинвариантность означает, что левые сдвиги являются аффинными преобразованиями в рассматриваемых аффинных координатах. Таким образом, в дальнейшем будем рассматривать

лоренцевы разрешимые группы Ли, т.е. группы Ли, оснащенные левоинвариантной лоренцевой метрикой.

В каком случае группа Ли T допускает левоинвариантную аффинную структуру? Согласно гипотезе Дж. Милнора [8], таковой является любая разрешимая группа Ли. С. Ямагучи установил, что полную левоинвариантную аффинную структуру допускают разрешимые группы Ли размерности $\leqslant 4$ [7].

2. Основные определения и обозначения

Пусть G_3 - связная односвязная 3-х-мерная разрешимая группа Ли, снабженная полной левоинвариантной аффинной структурой.

Определение 5. Аффинный причинный конус K_x в G_3 есть объединение всех лучей с началом $x \in G_3$, с направляющими векторами ξ, принадлежащими одной половине касательного конуса $\{\eta \in T_xG_3 \colon g_x(\eta,\eta) \geq 0\}$, где g –левоинваринтная лоренцева метрика на G_3, T_xG_3 - касательное пространство к G_3 в точке x.

Определение 6. Пусть $\mathcal{P} = \{P_x, x \in G_3\}$ семейство подмножеств G_3. Говорим, что $\mathcal{P}$ задает инвариантный порядок на группе G_3, если выполнены следующие условия:

P I: $x \in P_x$;

P II : если $y \in P_x$, то $P_y \subset P_x$;

P III : если $x \neq y$, то $P_x \neq P_y$;

P IV : если $L_x \colon G_3 \to G_3$ левый сдвиг, то $L_x(P_y) = P_{L_x(y)}$ для любого $x \in G_3$.

Определение 7. Порядок $\mathcal{P}$ на G_3 назовем аффинным причинным, если он задается семейством конусов $\{P_x, x \in G_3\}$ в G_3 таким, что P_x есть аффинный причинный конус в полной аффинной структуре, заданной на G_3.

Определение 8. Биекция $f \colon G_3 \to G_3$ называется порядковым автоморфизмом, если для любого $x \in G_3$ $f(P_x) = P_{f(x)}$.
Группу непрерывных порядковых автоморфизмов группы G_3 обозначим $Aut(\mathcal{P})$.

Пусть $Aut(\mathcal{P})_e$ - группа непрерывных порядковых автоморфизмов группы G_3 относительно порядка $\mathcal{P}$, оставляющих единицу группы e неподвижной.

Определение 9. Группа $Aut(\mathcal{P})_e$ действует транзитивно на множестве $B \subset G_3$, если для любых $x, y \in B$ существует автоморфизм $f \in Aut(\mathcal{P})_e$ такой, что $f(x) = y$.

Определение 10. Порядок $\mathcal{P}$ называется

1) внутренне однородным или int - однородным, если группа $Aut(\mathcal{P})_e$ действует транзитивно на $int(P_e)$;

2) гранично однородным или ∂ - однородным, если группа $Aut(\mathcal{P})_e$ действует транзитивно на $\partial P_e \setminus \{e\}$;

3) внешне однородным или ext - однородным, если группа $Aut(\mathcal{P})_e$ действует транзитивно на $G_3 \setminus (P_e \cup P_e^-)$.
Здесь $int(B)$ - внутренность множества B, ∂B - граница множества B,
$P_x^- = \{y \in G_3 \colon x \in P_y\}$.

Определение 11. Гладкое отображение $f \colon G_3 \to G_3$ сохраняет изотропные векторы, если $g_x(\xi,\xi) = 0 \Leftrightarrow g_{f(x)}(df_x(\xi), df_x(\xi)) = 0$, где $\xi \in T_xG_3$, T_xG_3 , - касательное пространство к G_3 в точке x, df_x - дифференциал отображения f, $df_x \colon T_xG_3 \to T_{f(x)}G_3$.

3. Однородные аффинные причинные порядки на разрешимых группах Ли G_3

Пусть G_3 - односвязная разрешимая группа Ли, $\mathcal{G}_3$ - ее алгебра Ли.

Известная классификация Бианки трехмерных вещественных алгебр Ли содержит девять типов, из них с I по VII охватывают все разрешимые алгебры, а типы VIII - IX - неразрешимые. Каждый из типов I - VII содержит коммутативный идеал $\mathcal{I}$ коразмерности 1.

Поскольку каждая левоинвариантная метрика g на группе Ли G_3 полностью задается невырожденной билинейной симметричной формой $a \colon \mathcal{G}_3 \times \mathcal{G}_3 \to R$ на алгебре Ли $\mathcal{G}_3$, то все левоинвариантные метрики на односвязной группе Ли с абелевой нормальной подгруппой N коразмерности 1 можно разбить на два больших класса в зависимости от ранга формы a на идеале $\mathcal{I}$ коразмерности 1, отвечающем N, а именно: класс 1, когда rank $a_{/\mathcal{I}} = 2$, т.е. форма a не вырождена на $\mathcal{I}$, и класс 2, когда rank $a_{/\mathcal{I}} = 1$, т.е форма вырождена на $\mathcal{I}$ (через rank $a_{/\mathcal{I}}$ обозначается ранг формы на идеале $\mathcal{I}$) [5]. С.П. Гаврилов предлагает упростить компоненты формы на $G_{\alpha\beta}$ формы a на $\mathcal{G}_3$ автоморфизмами алгебры Ли $\mathcal{G}_3$ [5]. Он получил, что для формы a класса 1 матрицу формы всегда можно записать в виде

$$(a) = (G_{\alpha\beta}) = \begin{pmatrix} G_{11} & G_{12} & 0 \\ G_{12} & G_{22} & 0 \\ 0 & 0 & G_{33} \end{pmatrix}.$$

Здесь $G_{33} \neq 0,\ G_{11}G_{22} - {G_{12}}^2 \neq 0$.

Затем вводится полная левоинвариантная аффинная структура, в которой левоинвариантные лоренцевы метрики из класса 1, в некоторой глобальной аффинной карте на G_3 имеют вид

$$g(x) = \begin{pmatrix} g_{11}(x^3) & g_{12}(x^3) & 0 \\ g_{12}(x^3) & g_{22}(x^3) & 0 \\ 0 & 0 & G_{33} \end{pmatrix}.$$

В данной глобальной аффинной карте $e = (0,0,0)$.

Для форм a класса 2, либо с помощью автоморфизмов алгебры Ли $\mathcal{G}_3$, либо путем преобразования идеала $\mathcal{I} = \langle e_1, e_2 \rangle$ и выбора трансверсального к $\mathcal{I}$ вектора e_3 всегда можно привести матрицу формы к виду

$$(a) = \begin{pmatrix} 0 & 0 & G_{13} \\ 0 & G_{22} & 0 \\ G_{13} & 0 & 0 \end{pmatrix}.$$

После этого на G_3 вводится полная левоинвариантная структура, в которой левоинвариантные метрики из класса 2 в некоторой глобальной аффинной карте на G_3 ($e = (0,0,0)$) имеют вид

$$g(x) = \begin{pmatrix} g_{11}(x^3) & g_{12}(x^3) & g_{13}(x^3) \\ g_{12}(x^3) & g_{22}(x^3) & g_{23}(x^3) \\ g_{13}(x^3) & g_{23}(x^3) & 0 \end{pmatrix}.$$

Более того, максимальная абелева подгруппа в G_3, в данных координатах, задается уравнением $x^3 = 0$.

Описанную выше полную левоинвариантную аффинную структуру на G_3 назовем канонической.

Для краткости формулировок будем называть разрешимые группы Ли с левоинвариантной метрикой класса 1 (класса 2) разрешимыми группами Ли класса 1 (класса 2).

Пусть семейство $\mathcal{P} = \{P_x\}$ задает на G_3 аффинный причинный порядок относительно канонической аффинной структуры. Нас интересует теперь задача вычисления группы порядковых автоморфизмов $Aut(\mathcal{P})$ для группы Ли G_3. Верна следующая

Теорема 1. *Пусть порядок на связной односвязной разрешимой группе G_3 такой, что P_x есть эллиптический конус в канонической аффинной структуре группы G_3. Тогда любой автоморфизм $f \in Aut(\mathcal{P})$ является аффинным преобразованием. [2]*

Аффинный порядковый автоморфизм $f\colon G_3 \to G_3$ в таком случае сохраняет изотропные векторы. Но тогда f, как известно, будет конформным преобразованием. Для группы Ли конформных преобразований вида

$$f^i = \sum_{k=1}^{3} a^i_k x^k + \alpha^i$$

векторы Киллинга имеют вид

$$\xi^i = \sum_{k=1}^{3} b^i_k x^k + \beta^i.$$

Напомним также, что векторы Киллинга должны удовлетворять уравнениям

$$\xi^n \frac{\partial g_{ik}}{\partial x^n} + g_{in}\frac{\partial \xi^n}{\partial x^k} + g_{kn}\frac{\partial \xi^n}{\partial x^i} = \lambda g_{ik}.$$

Если $\lambda = 0$, то f - изометрия; если $\lambda = const$, то f - гомотетия; если $\lambda = \lambda(x)$, где $\lambda(x)$ - некоторая функция x, то f - конформное преобразование.

Группу изометрий группы Ли G_3 обозначим $Isom(G_3)$, группу гомотетий - $Hom(G_3)$, группу конформных преобразований - $Conf(G_3)$. Ясно, что $Isom(G_3) \subset Hom(G_3) \subset Conf(G_3)$.

Далее будут использоваться следующие факты.

Предложение 1. *Пусть G_3 – связная односвязная разрешимая группа Ли. На G_3 задан аффинный причинный порядок $\mathcal{P}$ относительно канонической аффинной структуры $f \in Aff(G_3) \cap Aut(\mathcal{P})$, то $f \in Aut(\mathcal{P})_e \Leftrightarrow f^i = \sum\limits_{k=1}^{3} a_k^i x^k$.*

Доказательство. Очевидно. ■

Предложение 2. *Пусть группа Ли G_3 и $\mathcal{P} = \{P_x\}$ порядок, заданный на ней, удовлетворяют условиям Предложения 1. Если группа $Aut(\mathcal{P})_e$ состоит из порядковых автоморфизмов вида $f^j = \sum\limits_{k=1}^{3} a_k^j x^k$, $j = 1, 2$, $f^3 = x^3$, то порядок $\mathcal{P}$ не будет ни int - однородным, ни ∂ - однородным, ни ext - однородным.*

Доказательство. Пусть $f^j = \sum\limits_{k=1}^{3} a_k^j x^k$, $j = 1, 2$, $f^3 = x^3$ и $H_\alpha = \{x^3 = \alpha\}$. На G_3 задано семейство эллиптических конусов $\{P_x\}$. Очевидно: $f^3(H_\alpha) = H_\alpha$. При некотором выборе α_1, α_2, α_3 получим $H_{\alpha_1} \cap intP_e \neq \varnothing$; $H_{\alpha_2} \cap \partial P_e \setminus \{e\} \neq \varnothing$; $H_{\alpha_3} \cap G_3 \setminus (P_e \cup P_e^-) \neq \varnothing$. Поэтому $f \in Aut(\mathcal{P})_e$ не будет действовать транзитивно ни на $int(P_e)$, ни на $\partial P_e \setminus \{e\}$, ни на $G_3 \setminus (P_e \cup P_e^-)$. По определению 10 это означает, что порядок $\mathcal{P}$ не будет ни $int -$, $\partial -$, $ext -$ однородным. ■

Возникает вопрос: для каких групп Ли G_3 порядок $\mathcal{P}$ будет $int -$ однородным, $\partial -$ однородным, $ext -$ однородным? Ответ дает

Теорема 2. *Пусть G_3 – связная односвязная разрешимая группа Ли с левоинвариантной лоренцевой метрикой g. Пусть на G_3 задан аффинный причинный порядок $\mathcal{P}$ относительно канонической аффинной структуры. Тогда на группах Ли G_3 типов I, VI_0, VII_0 класса 1 существующий порядок будет $int -$ однородным, $\partial -$ однородным, $ext -$ однородным (при этом метрика g - плоская). В остальных случаях порядок не является однородным ни в одном из указанных смыслов.*

Как было замечено выше, аффинное преобразование, сохраняющее изотропные векторы, будет конформным.

Верна следующая

Теорема 3. *Пусть H – группа Ли аффинных преобразований относительно канонической аффинной структуры, сохраняющих изотропные векторы левоинвариантной лоренцевой метрики g на G_3. Тогда*

1) для группы Ли G_3I $H \subset Hom(G_3)$;

2) для групп Ли $G_3II - G_3VII$ класса 1 $H \subset Isom(G_3)$, за исключением того случая, когда метрика плоская; в этом случае $H \subset Hom(G_3)$;

3) для групп Ли G_3II, G_3III, G_3V, G_3VII класса 2 $H \subset Hom(G_3)$;

4) для группы Ли G_3IV класса 2 $H \subset Isom(G_3)$;

5) для группы Ли G_3VI $H \subset Hom(G_3)$, за исключением того случая, когда метрика имеет вид $g_{ij}(x^3) = e^{-x^3}a_{ij}$, где $a_{ij} = const$. В этом случае $H \subset Conf(G_3)$.

Аналогичная теорема приведена в [2] для $f\colon G_3 \to G_3$, являющегося аффинным преобразованием относительно некоторой аффинной структуры на связной односвязной группе Ли G_3.

4. Доказательство Теоремы 2 и Теоремы 3

4.1. Доказательство Теоремы 2

Изложим краткую схему доказательства.

Нам надо найти группу $Aut(\mathcal{P})_e$ для группы G_3. Используя Предложение 1, замечаем, что $f \in Aut(\mathcal{P})_e \Leftrightarrow f^i = \sum\limits_{k=1}^{3} a_k^i x^k$. Для преобразований такого вида векторы Киллинга имеют вид $\xi^i = \sum\limits_{k=1}^{3} b_k^i x^k$. Обозначим через $b_j^i = \dfrac{\partial \xi^i}{\partial x^j}$.

В ходе доказательства используются виды левоинвариантных метрик, полученные С.П. Гавриловым [5]. Компоненты метрики g на группе G_3 зависят только от x^3 (для любого класса). Поэтому уравнения Киллинга для векторов ξ^i имеют вид

$$b_s^3 \frac{\partial g_{ik}}{\partial x^3} + g_{in}b_k^n \ + g_{kn}b_i^n = \lambda g_{ik}. \tag{1}$$

Интегрируя уравнения (1), находим векторы Киллинга $\xi_{(n)}^i$, $n = 1, \ldots, r$, где r - размерность $Aut(\mathcal{P})_e$. Далее используется метод, изложенный в [6, с. 181]. Идея этого метода заключается в следующем. Интегрируя систему

$$\begin{cases} \dfrac{dx^i}{dt_n} = \xi_{(n)}^i(x), \\ \quad x^i(0) = x_0^i, \end{cases} \tag{2}$$

получаем x^i как функцию от t_n и x_0^i, т.е. $x^i = h(t_n, x_0^i)$. Затем, делая замену x^i на $\bar{x}^i$ и x_0^i на x^i, имеем окончательно $x^i = h(t_n, x^i)$. Это однопараметрическая группа преобразований из $Aut(\mathcal{P})_e$, соответствующая $\xi_{(n)}^i$. Теперь останется проверить действие группы $Aut(\mathcal{P})_e$ на $int(P_e)$, $\partial P_e \setminus \{e\}$, $G_3 \setminus (P_e \cup P_e^-)$.

Перейдем к подробному доказательству.

Сначала будем рассматривать группы G_3 класса 1.

1. G_3I

Метрика имеет вид $g_{11} = V_1$, $g_{22} = V_2$, $g_{33} = V_3$, $V_i = 1$, $i = 1, 2, 3$. Известно, что на группе Ли G_3I аффинный причинный порядок будет int − однородным, ∂ − однородным, ext − однородным.

2. G_3III

Метрика имеет вид $g_{11} = V_1 e^{-2x^3}$, $g_{22} = V_2$, $g_{33} = G_{33}$. Выпишем уравнения Киллинга (1)

$$
\begin{gathered}
-b_i^3 x^i V_1 + V_1 b_1^1 = \frac{\lambda}{2} V_1, \\
V_1 e^{-2x^3} b_2^1 + V_2 b_1^2 = 0, \\
V_1 e^{-2x^3} b_3^1 + G_{33} b_1^3 = 0, \\
V_2 b_2^2 = \frac{\lambda}{2} V_2, \\
V_2 b_3^2 + G_{33} b_2^3 = 0, \\
b_3^3 = \frac{\lambda}{2}.
\end{gathered}
$$

Нетрудно заметить, что они выполняются, если $b_1^1 = 0$, $b_1^2 = 0$, $b_1^3 = 0$, $b_2^1 = 0$, $b_2^2 = 0$, $b_2^3 = 0$, $b_3^1 = 0$, $b_3^2 = 0$, $b_3^3 = 0$, $\lambda = 0$. Следовательно, $\xi^i = (0, 0, 0)$. Интегрируя систему (2), получаем

$$
\begin{cases}
\bar{x}^1 = x^1, \\
\bar{x}^2 = x^2, \\
\bar{x}^3 = x^3.
\end{cases}
$$

Откуда видно, что $Aut(\mathcal{P})_e$ состоит только из тождественного преобразования. В таком случае, согласно Предложению 2, аффинный причинный порядок не будет ни $int\ -$, ни $\partial\ -$, ни $ext\ -$ однородным.

3. G_3IV

Метрика имеет вид $g_{11} = V_1 e^{-2x^3}$, $g_{12} = -V_1 x^3 e^{-2x^3}$, $g_{22} = (G_{22} + V_2 (x^3)^2) e^{-2x^3}$, $g_{33} = G_{33}$.

Повторяя аналогичные вычисления и рассуждения, делаем вывод, что аффинный причинный порядок не является $int\ -$, $\partial\ -$, $ext\ -$ однородным и $\lambda = 0$.

4. G_3V

Метрика имеет вид $g_{11} = V_1 e^{-2x^3}$, $g_{22} = V_2 e^{-2x^3}$, $g_{33} = G_{33}$.

Запишем уравнения Киллинга:

$$-b_i^3 x^i + b_1^1 = \frac{\lambda}{2},$$
$$V_1 b_2^1 + V_2 b_1^2 = 0,$$
$$V_1 e^{-2x^3} b_3^1 + G_{33} b_1^3 = 0,$$
$$-b_i^3 x^i + b_2^2 = \frac{\lambda}{2},$$
$$V_2 2e^{-2x^3} b_3^2 + G_{33} b_2^3 = 0,$$
$$b_3^3 = \frac{\lambda}{2}.$$

Откуда получаем: $b_1^1 = 0$, $b_i^3 = 0$, $b_3^1 = 0$, $b_2^2 = 0$, $\lambda = 0$, $b_1^2 = -\frac{V_1}{V_2} b_2^1$. Тогда $\xi^i = (b_2^1 x^2, -\frac{V_1}{V_2} b_2^1 x^1, 0)$. В этом случае вектор Киллинга имеет вид $\xi^i_{(1)} = \left(x^2, -\frac{V_1}{V_2} x^1, 0\right)$. Рассмотрим систему (2). В зависимости от знака отношения $\frac{V_1}{V_2}$ возможны два варианта.

1) $\frac{V_1}{V_2} = -1$, тогда

$$\begin{cases} \bar{x}^1 = x^1 \operatorname{ch} t + x^2 \operatorname{sh} t, \\ \bar{x}^2 = x^1 \operatorname{sh} t + x^2 \operatorname{ch} t, \\ \bar{x}^3 = x^3. \end{cases}$$

2) $\frac{V_1}{V_2} = 1$, имеем

$$\begin{cases} \bar{x}^1 = x^1 \cos t + x^2 \sin t, \\ \bar{x}^2 = -x^1 \sin t + x^2 \cos t, \\ \bar{x}^3 = x^3. \end{cases}$$

Группа $Aut(\mathcal{P})_e$ (в зависимости от знака $\frac{V_1}{V_2}$) состоит из полученных преобразований. И в том, и в другом случае, используя Предложение 2, получаем, что порядок не будет $int -$, $\partial -$, $ext -$ однородным.

5. $G_3 VI$

Метрика имеет вид $g_{11} = V_1 e^{-2x^3}$, $g_{12} = G_{12} e^{-(1+\alpha)x^3}$, $g_{22} = V_2 e^{-2\alpha x^3}$, $g_{33} = G_{33}$, $\alpha \in (-1, 0) \cup (0, 1)$. Повторяя аналогичные вычисления и рассуждения, получаем, что $Aut(\mathcal{P})_e$ состоит только из тождественного преобразования и, следовательно, порядок не будет $int -$, $\partial -$, $ext -$ однородным.

6. G_3VI_0

6.1. $G_3VI_{0_1}$

Метрика имеет вид $g_{11} = V_1e^{-2x^3}$, $g_{12} = G_{12}$, $g_{22} = V_2e^{-2x^3}$, $g_{33} = G_{33}$. После вычислений получаем $\xi^i = (0, 0, 0)$.

Значит, аффинный порядок не является $int -$, $\partial -$, $ext -$ однородным и $\lambda = 0$.

6.2. $G_3VI_{0_2}$

Метрика имеет вид $g_{12} = 1$, $g_{33} = G_{33}$. Выпишем уравнения Киллинга

$$\begin{gathered} b_1^2 = 0, \\ b_2^2 + b_1^1 = \lambda, \\ b_3^2 + G_{33}b_1^3 = 0, \\ b_2^1 = 0, \\ b_3^1 + G_{33}b_2^3 = 0, \\ b_3^3 = \frac{\lambda}{2}. \end{gathered}$$

Следовательно, $b_2^2 = \lambda - b_1^1$, $b_1^3 = -\frac{1}{G_{33}}b_3^2$ и $b_2^3 = -\frac{1}{G_{33}}b_3^1$.

Тогда $\xi^i = (b_1^1x^1 + b_3^1x^3,\ (\lambda - b_1^1)x^2 + b_3^2x,\ -\frac{1}{G_{33}}b_3^2x^1 - \frac{1}{G_{33}}b_3^1x^2 + \frac{\lambda}{2}x^3)$.

Запишем векторы Киллинга:

$\xi^i_{(1)} = (x^1,\ -x^2,\ 0)$, $\xi^i_{(2)} = (x^3,\ 0, -\frac{1}{G_{33}}x^2)$, $\xi^i_{(3)} = (0,\ x^3, -\frac{1}{G_{33}}x^1)$,

$\xi^i_{(4)} = (0,\ x^2,\ \frac{1}{2}x^3)$.

Интегрируя систему (2) для каждого $\xi^i_{(n)}$, $n = 1,\ 2,\ 3,\ 4$, получаем

$$\begin{cases} \bar{x}^1 = x^1e^{\varphi}, \\ \bar{x}^2 = x^2e^{-\varphi}, \\ \bar{x}^3 = x^3; \end{cases}$$

$$\begin{cases} \bar{x}^1 = x^1 + x^3t - \dfrac{1}{2G_{33}}x^2t^2, \\ \bar{x}^2 = x^2, \\ \bar{x}^3 = x^3 - \dfrac{1}{G_{33}}x^2t; \end{cases}$$

$$\begin{cases} \bar{x}^1 = x^1, \\ \bar{x}^2 = x^2 + x^3\mu - \dfrac{1}{2G_{33}}x^1\mu^2, \\ \bar{x}^3 = x^3 - \dfrac{1}{G_{33}}x^1\mu; \end{cases}$$

$$\begin{cases} \bar{x}^1 = x^1, \\ \bar{x}^2 = x^2 e^{\phi}, \\ \bar{x}^3 = x^3 e^{\frac{\phi}{2}}. \end{cases}$$

Группа $Aut(\mathcal{P})_e$ состоит из композиций полученных преобразований. Ясно, что $Aut(\mathcal{P})_e$ будет действовать транзитивно на $int(P_e)$, $\partial P_e \setminus \{e\}$, $G_3 \setminus (P_e \cup P_e^-)$. Следовательно, на группе $G_3 V I_{0_2}$ порядок является int – однородным, ∂ – однородным, ext – однородным одновременно (причем метрика плоская).

7. $G_3 V II$

Выпишем метрику
$g_{11} = \frac{1}{2} e^{-2\alpha x^3}(V_1 + G_{22} + (V_1 - G_{22})\cos 2\beta x^3)$,
$g_{12} = \frac{1}{2} e^{-2\alpha x^3}(V_1 - G_{22})\sin 2\beta x^3$,
$g_{22} = \frac{1}{2} e^{-2\alpha x^3}(V_1 + G_{22} - (V_1 - G_{22})\cos 2\beta x^3)$,
$g_{33} = G_{33}$, $0 < \beta < 1$, $\alpha = \sqrt{1-\beta^2}$.
Обозначим $V_1 + G_{22} = B$, $V_1 - G_{22} = C$.
Тогда уравнения Киллинга имеют вид

$$-b_i^3 x^i(\alpha(B + C\cos 2\beta x^3) + \beta C \sin 2\beta x^3) + (B + Cb_1^1 \cos 2\beta x^3) + Cb_1^2 \sin 2\beta x^3 = \frac{\lambda}{2}(B + C\cos 2\beta x^3),$$

$$-2b_i^3 x^i(\alpha C \sin 2\beta x^3 - \beta C \cos 2\beta x^3) + (B + C\cos 2\beta x^3)b_2^1 + C(b_1^1 + b_2^2)\sin 2\beta x^3 + (B - C\cos 2\beta x^3)b_1^2 = \lambda C \sin 2\beta x^3,$$

$$(B + C\cos 2\beta x^3)b_3^1 + Cb_3^2 \sin 2\beta x^3 + 2G_{33} e^{2\alpha x^3} b_1^3 = 0,$$

$$-b_i^3 x^i(\alpha(B - C\cos 2\beta x^3) - \beta C \sin 2\beta x^3) + Cb_2^1 \sin 2\beta x^3 + (B - C\cos 2\beta x^3)b_2^2 = \frac{\lambda}{2}(B - C\cos 2\beta x^3),$$

$$Cb_3^1 \sin 2\beta x^3 + (B - C\cos 2\beta x^3)b_3^2 + 2G_{33} e^{2\alpha x^3} b_2^3 = 0,$$

$$b_3^3 = \frac{\lambda}{2}.$$

Получаем два возможных случая:
А) $C = 0,\ b_3^1 = 0,\ b_1^3 = 0$;
В) $B = 0,\ b_3^1 = 0,\ b_1^3 = 0,\ b_3^2 = 0$.

Рассмотрим случай А).

Имеем: $C = 0,\ b_3^1 = 0,\ b_1^3 = 0$. Нетрудно заметить, что $b_3^2 = 0, b_2^3 = 0, b_2^1 = -b_1^2$, $b_3^3 = 0,\ \lambda = 0,\ b_1^1 = 0,\ b_2^2 = 0$. Окончательно получаем $\xi^i = (b_2^1 x^2,\ -b_2^1 x^1,\ 0)$. Вектор Киллинга имеет вид $\xi^i_{(1)} = (x^2,\ -x^1,\ 0)$. Интегрируя систему (2), получаем

$$\begin{cases} \bar{x}^1 = x^1 \cos t + x^2 \sin t, \\ \bar{x}^2 = -x^1 \sin t + x^2 \cos t, \\ \bar{x}^3 = x^3. \end{cases}$$

Группа $Aut(\mathcal{P})_e$ состоит из преобразований данного вида. Эти преобразования подходят под условия Предложения 2. Следовательно, порядок не будет $int-$, $\partial-$, $ext-$ однородным.

Рассмотрим случай В).

Имеем: $B = 0,\ b_3^1 = 0,\ b_1^3 = 0,\ b_3^2 = 0$. Находим $b_2^3 = 0,\ b_3^3 = 0,\ \lambda = 0$, $b_1^1 = 0,\ b_1^2 = 0, b_2^2 = 0,\ b_2^1 = 0$.
В итоге получаем $\xi^i = (0, 0, 0)$. Вывод: порядок не будет $int-$, $\partial-$, $ext-$ однородным.

8. G_3VII_0

Метрика имеет вид

$$g_{11} = \frac{1}{2}(V_1 + G_{22} + (V_1 - G_{22})\cos 2x^3),$$

$$g_{12} = \frac{1}{2}(V_1 - G_{22})\sin 2x^3,$$

$$g_{22} = \frac{1}{2}(V_1 + G_{22} - (V_1 - G_{22})\cos 2x^3),$$

$$g_{33} = G_{33}.$$

Обозначим $V_1 + G_{22} = B$, $V_1 - G_{22} = C$.
Выпишем уравнения Киллинга:

$$-b_i^3 x^i C \sin 2x^3 + (B + C\cos 2x^3)b_1^1 + Cb_1^2 \sin 2x^3 = \frac{\lambda}{2}(B + C\cos 2x^3),$$

$$2b_i^3 x^i C\cos 2x^3 + (B + C\cos 2x^3)b_2^1 + C(b_1^1 + b_2^2)\sin 2x^3 + \\ +(B - C\cos 2x^3)b_1^2 = \lambda(C\sin 2x^3),$$

$$(B + C\cos 2x^3)b_3^1 + Cb_3^2 \sin 2x^3 + 2G_{33}b_1^3 = 0,$$

$$b_i^3 x^i C\sin 2x^3 + Cb_2^1 \sin 2x^3 + (B - C\cos 2x^3)b_2^2 = \frac{\lambda}{2}(B - C\cos 2x^3),$$

$$Cb_3^1 \sin 2x^3 + (B - C\cos 2x^3)b_3^2 + 2G_{33}b_2^3 = 0,$$

$$b_3^3 = \frac{\lambda}{2}.$$

Уравнения выполняются при условии, что $Cb_3^1 = 0,\ Bb_3^1 + 2G_{33}b_1^3 = 0,$ $Cb_3^2 = 0$. Это возможно, если:
А) $b_3^1 = 0,\ b_1^3 = 0,\ b_3^2 = 0;$
В) $C = 0,\ b_1^3 = -b_3^1\dfrac{V_1}{G_{33}}.$

Рассмотрим случай А).
Имеем: $b_3^1 = 0,\ b_1^3 = 0,\ b_3^2 = 0$. Следовательно, $b_1^1 = 0,\ b_1^2 = 0,\ b_2^1 = 0,\ b_2^2 = 0,\ b_2^3 = 0,\ b_3^3 = 0$ и $\lambda = 0$. Значит, $\xi^i = (0,0,0)$. Порядок не будет $int -$, $\partial -$, $ext -$ однородным.
Рассмотрим случай В).
$C = 0,\ b_1^3 = -b_3^1\dfrac{V_1}{G_{33}}$. Имеем: $b_1^1 = \dfrac{\lambda}{2},\ b_2^1 = -b_1^2,\ b_2^2 = \dfrac{\lambda}{2},\ \ b_2^3 = -\dfrac{V_1}{G_{33}}b_3^2,\ b_3^3 = \dfrac{\lambda}{2}.$
Обозначим $D = \sqrt{-\dfrac{V_1}{G_{33}}}$. Тогда
$\xi^i = (\dfrac{\lambda}{2}x^1 + b_2^1x^2 + b_3^1x^3,\ -b_2^1x^1 + \dfrac{\lambda}{2}x^2 + b_3^2x^3,\ D^2b_3^1x^1 + D^2b_3^2x^2 + \dfrac{\lambda}{2}x^3).$
Векторы Киллинга имеют вид $\xi^i_{(1)} = (x^1,\ x^2,\ x^3),\ \xi^i_{(2)} = (x^2,\ -x^1,\ 0),$ $\xi^i_{(3)} = (x^3,\ 0,\ D^2x^1),\ \xi^i_{(4)} = (0,\ x^3,\ D^2x^2)$. Интегрируя систему (2) для каждого $\xi^i_{(n)},\ n = 1,\ 2,\ 3,\ 4$, получаем

$$\begin{cases} \bar{x}^1 = x^1e^t, \\ \bar{x}^2 = x^2e^t, \\ \bar{x}^3 = x^3e^t; \end{cases}$$

$$\begin{cases} \bar{x}^1 = x^1\cos\mu + x^2\sin\mu, \\ \bar{x}^2 = -x^1\sin\mu + x^2\cos\mu, \\ \bar{x}^3 = x^3; \end{cases}$$

$$\begin{cases} \bar{x}^1 = x^1\operatorname{ch}D\varphi + \dfrac{x^3}{D}\operatorname{sh}D\varphi, \\ \bar{x}^2 = x^2, \\ \bar{x}^3 = D(x^1\operatorname{sh}D\varphi + \dfrac{x^3}{D}\operatorname{ch}D\varphi); \end{cases}$$

$$\begin{cases} \bar{x}^1 = x^1, \\ \bar{x}^2 = x^2\operatorname{ch}D\psi + \dfrac{x^3}{D}\operatorname{sh}D\psi, \\ \bar{x}^3 = D(x^2\operatorname{sh}D\psi + \dfrac{x^3}{D}\operatorname{ch}D\psi). \end{cases}$$

Группа $Aut(\mathcal{P})_e$ состоит из группы Лоренца и подобий. Известно, что группа Лоренца действует транзитивно на $int(P_e)$, $\partial P_e \setminus \{e\}$, $G_3 \setminus (P_e \cup P_e^-)$. Следовательно, на группе G_3VII_0 в случае, когда $G_{22} = V_1$, аффинный

причинный порядок является $int-$, $\partial-$, $ext-$ однородным.

Теперь рассмотрим разрешимые группы Ли G_3 класса 2.

9. G_3III

Метрика имеет вид $g_{13}=1$, $g_{22}=V_2e^{-2x^3}$.
Запишем уравнения Киллинга:

$$\begin{gathered}2b_1^3=0,\\ V_2e^{-2x^3}b_1^2+b_2^3=0,\\ b_3^3+b_1^1=\lambda,\\ -b_i^3x^i+b_2^2=\frac{\lambda}{2},\\ V_2e^{-2x^3}b_3^2+b_2^1=0,\\ 2b_3^1=0.\end{gathered}$$

Решая уравнения, получаем: $b_1^1=\lambda$, $b_1^2=0$, $b_1^3=0$, $b_2^1=0$, $b_2^2=\frac{\lambda}{2}$, $b_3^1=0$, $b_3^2=0$, $b_3^3=0$ и $\lambda=const$. Тогда окончательно имеем, что $\xi^i=(\lambda x^1,\ \frac{\lambda}{2}x^2,\ 0)$. Вектор Киллинга будет иметь следующий вид $\xi^i_{(1)}=(x^1,\ \frac{1}{2}x^2,\ 0)$. Интегрируя систему (2), получаем

$$\begin{cases}\bar{x}^1=x^1e^t,\\ \bar{x}^2=x^2e^{\frac{t}{2}},\\ \bar{x}^3=x^3.\end{cases}$$

Группа $Aut(\mathcal{P})_e$ состоит из полученных преобразований. Используя Предложение 2 можно сделать вывод, что аффинный причинный порядок не является $int-$, $\partial-$, $ext-$ однородным.

10. G_3V

Метрика имеет вид $g_{13}=e^{-x^3}$, $g_{22}=V_2e^{-2x^3}$.
Рассмотрим уравнения Киллинга:

$$\begin{gathered}2e^{-x^3}b_1^3=0,\\ e^{-x^3}b_2^3+V_2e^{-2x^3}b_1^2=0,\\ -b_i^3x^i+b_1^1+b_3^3=\lambda,\\ -b_i^3x^i+b_2^2=\frac{\lambda}{2},\\ V_2e^{-2x^3}b_3^2+e^{-x^3}b_2^1=0,\\ 2e^{-x^3}b_3^1=0.\end{gathered}$$

Следовательно, $b_1^1 = \lambda$, $b_1^2 = 0$, $b_1^3 = 0$, $b_2^1 = 0$, $b_2^2 = \frac{\lambda}{2}$, $b_3^1 = 0$, $b_3^2 = 0$, $b_3^3 = 0$ и $\lambda = const$. Тогда окончательно имеем, что $\xi^i = (\lambda x^1,\ \frac{\lambda}{2}x^2,\ 0)$. Повторяя рассуждения, получаем, что аффинный причинный порядок не является является $int -$, $\partial -$, $ext -$ однородным.

Теорема 2 доказана.

В остальных связных односвязных разрешимых группах Ли аффинный причинный порядок не существует [1].

4.2. Доказательство теоремы 3

Как известно, аффинное преобразование, сохраняющее изотропные векторы, будет конформным. Для группы Ли конформных преобразований вида

$$f^i = \sum_{k=1}^{3} a_k^i x^k + \alpha^i$$

векторы Киллинга имеют вид

$$\xi^i = \sum_{k=1}^{3} b_k^i x^k + \beta^i.$$

Обозначим $b_j^m = \dfrac{\partial \xi^m}{\partial x^3}$. Напомним, что метрика на группах Ли зависит только от x^3. Тогда уравнения Киллинга для векторов ξ^i можно записать в следующем виде:

$$(b_s^3 x^s + \beta^3)\frac{\partial g_{ik}}{\partial x^3} + g_{in}b_k^n\ + g_{km}b_i^m = \lambda g_{ik}.$$

Для доказательства теоремы надо найти λ. Если $\lambda = 0$, то $H \subset Isom(G_3)$; если $\lambda = const$, то $H \subset Hom(G_3)$; если $\lambda = \lambda(x)$, где $\lambda(x)$ - некоторая функция x, то $H \subset Conf(G_3)$.

Сначала рассмотрим группы Ли G_3 класса 1.
Левоинвариантные метрики на группах Ли G_3 класса 1 имеют вид:

$$g(x) = \begin{pmatrix} g_{11}(x^3) & g_{12}(x^3) & 0 \\ g_{12}(x^3) & g_{22}(x^3) & 0 \\ 0 & 0 & G_{33} \end{pmatrix}.$$

Выпишем уравнение Киллинга для пары (ik) равной (33): $b_3^3 = \dfrac{\lambda}{2}$. Следовательно, $\lambda = const$, и $H \subset Hom(G_3)$.

Рассмотрим группы, у которых метрика не является плоской.

1. G_3II

Метрика имеет вид $g_{11} = G_{11}$, $g_{12} = -G_{11}x^3$, $g_{22} = V_2 + G_{11}(x^3)^2$, $g_{33} = G_{33}$. Рассмотрим уравнения Киллинга , соответствующие парам (ik) : (11), (12), (22):

$$b_1^1 - x^3 b_1^2 = \frac{\lambda}{2},$$
$$-(b_i^3 x^i + \beta^3)G_{11} + G_{11}b_2^1 - G_{11}x^3(b_1^1 + b_2^2) + (V_2 + G_{11}x^{3^2})b_1^2 = -\lambda G_{11}x^3,$$
$$(b_i^3 x^i + \beta^3)G_{11}x^3 - G_{11}x^3 b_2^1 + (V_2 + G_{11}x^{3^2})b_2^2 = \lambda(V_2 + G_{11}x^{3^2}).$$

Решая уравнения, получаем, что $b_2^2 = \frac{\lambda}{2}$, и $b_2^2 = 0$, следовательно, $\lambda = 0$, т.е. $H \subset Isom(G_3II)$.

2. G_3III

2.1. G_3III_1

Метрика имеет вид $g_{11} = V_1 e^{-2x^3}$, $g_{12} = G_{12}e^{-x^3}$, $g_{22} = V_2$, $g_{33} = G_{33}$. Выпишем уравнения Киллинга для пары (ik) равной (11), (33)

$$-(b_i^3 x^i + \beta^3)V_1 + +V_1 b_1^1 + G_{12}e^{-x^3}b_1^3 = \frac{\lambda}{2}V_1,$$
$$b_3^3 = \frac{\lambda}{2}.$$

Нетрудно заметить, что $b_i^3 = 0$, $i = 1, 2, 3$. Тогда $\lambda = 0$, и $H \subset Isom(G_3III_1)$.

2.2. G_3III_2

Метрика имеет вид $g_{11} = V_1 e^{-2x^3}$, $g_{12} = e^{-x^3}$, $g_{33} = G_{33}$.

2.3. G_3III_3

Метрика имеет вид $g_{11} = e^{-x^3}$, $g_{22} = V_2$, $g_{33} = G_{33}$.

2.4. G_3III_4

Метрика имеет вид $g_{12} = e^{-x^3}$, $g_{33} = G_{33}$.

Повторяя аналогичные вычисления и рассуждения, получаем $\lambda = 0$ и $H \subset Isom(G_3III_2)$, $H \subset Isom(G_3III_3)$, $H \subset Isom(G_3III_4)$.

Вывод: для группы Ли G_3III $H \subset Isom(G_3)$.

3. G_3IV

3.1. G_3IV_1

Метрика имеет вид $g_{11} = V_1 e^{-2x^3}$, $g_{12} = -V_1 x^3 e^{-2x^3}$,
$g_{22} = (G_{22} + V_2(x^3)^2)e^{-2x^3}$, $g_{33} = G_{33}$.

В ходе доказательства Теоремы 2 получили, что $\lambda = 0$ и, следовательно, $H \subset Isom(G_3IV_1)$.

3.2. G_3IV_2

Метрика имеет вид $g_{12} = e^{-2x^3}$, $g_{22} = -2x^3e^{-2x^3}$, $g_{33} = G_{33}$.
Рассмотрим уравнения Киллинга, соответствующее парам (ik) : (11), (12), (33):

$$2e^{-2x^3}b_1^2 = 0,$$
$$-2(b_i^3x^i + \beta^3) + b_1^1 + b_2^2 - 2x^3b_1^2 = \lambda,$$
$$b_3^3 = \frac{\lambda}{2}.$$

Нетрудно заметить, что $b_1^2 = \frac{\lambda}{2}$, $b_i^3 = 0$, $i = 1, 2, 3$, следовательно, $\lambda = 0$, значит, $H \subset Isom(G_3IV_2)$.

Вывод: для группы Ли G_3IV $H \subset Isom(G_3IV)$.

4. G_3V

Метрика имеет вид $g_{11} = V_1e^{-2x^3}$, $g_{22} = V_1x^3e^{-2x^3}$, $g_{33} = G_{33}$.

В ходе доказательства Теоремы 2 получили, что $\lambda = 0$. Откуда следует, что $H \subset Isom(G_3V)$.

5. G_3VI

5.1. G_3VI_1

Метрика имеет вид $g_{11} = V_1e^{-2x^3}$, $g_{12} = G_{12}e^{-(1+\alpha)x^3}$, $g_{22} = V_2e^{-2x^3}$, $g_{33} = G_{33}$.

В ходе доказательства Теоремы 2 получили, что $\lambda = 0$ и, следовательно, $H \subset Isom(G_3VI_1)$.

5.2. G_3VI_2

Метрика имеет вид $g_{11} = V_1e^{-2x^3}$, $g_{12} = e^{-(1+\alpha)x^3}$, $g_{33} = G_{33}$.

5.3. G_3VI_3

Метрика имеет вид $g_{12} = e^{-(1+\alpha)x^3}$, $g_{22} = V_2e^{-2x^3}$, $g_{33} = G_{33}$.

Повторяя аналогичные вычисления, получаем $\lambda = 0$ и $H \subset Isom(G_3VI_2)$, $H \subset Isom(G_3VI_3)$.

Вывод: для группы Ли G_3VI класса 1 $H \subset Isom(G_3VI)$.

6. G_3VI_0

6.1. $G_3VI_{1_0}$

Метрика имеет вид $g_{11} = V_1e^{-2x^3}$, $g_{12} = G_{12}$, $g_{22} = V_2e^{2x^3}$, $g_{33} = G_{33}$.

В ходе доказательства Теоремы 2 получили $\lambda = 0$. Следовательно, $H \subset Isom(G_3VI_{1_0})$.

6.2. $G_3VI_{0_3}$

Метрика имеет вид $g_{11} = V_1e^{-2x^3}$, $g_{12} = 1$, $g_{33} = G_{33}$.

6.3. $G_3VI_{0_4}$

Метрика имеет вид $g_{12} = 1$, $g_{22} = V_2e^{2x^3}$, $g_{33} = G_{33}$.

Повторяя аналогичные вычисления, получаем $\lambda = 0$ и, следовательно, $H \subset Isom(G_3VI_{0_3})$, $H \subset Isom(G_3VI_{0_4})$.

Вывод: для группы Ли G_3VI_0 класса 1, если метрика не плоская, то $H \subset Isom(G_3VI_0)$.

7. G_3VII

В ходе доказательства Теоремы 2 получили $\lambda = 0$. Следовательно, $H \subset Isom(G_3VII)$.

8. G_3VII_0

В случае, когда метрика не плоская, в ходе доказательства Теоремы 2 получили, что $\lambda = 0$. Значит, $H \subset Isom(G_3VII_0)$.

Рассмотрим группы Ли класса 2.

9. G_3II

Метрика имеет вид $g_{13} = 1$, $g_{22} = V_2$, $g_{23} = -x^3$.
Выпишем уравнение Киллинга для пары (ik) равной (11), (13)

$$b_1^3 = 0,$$
$$b_1^1 + b_3^3 + V_2b_1^2 - x^3b_1^3 = \lambda.$$

Получаем, что $\lambda = const$, значит, $H \subset Hom(G_3II)$.

10. G_3III

10.1. G_3III_1

Метрика имеет вид $g_{11} = V_2(e^{-x^3} - 1)$, $g_{12} = e^{-x^3} - 1$, $g_{13} = G_{13}e^{-x^3}$, $g_{22} = V_2$.
Уравнение Киллинга для пары (ik) равной (11), (12), (33):

$$(e^{-x^3} - 1)b_2^1 + V_2 b_2^2 = \frac{\lambda}{2}V_2,$$
$$(e^{-x^3} - 1)b_3^1 + V_2 b_3^2 + G_{13}e^{-x^3}b_2^1 = 0,$$
$$2(e^{-x^3} - 1)b_3^1 = 0.$$

Следовательно, $\lambda = const$, и $H \subset Hom(G_3III_1)$.

10.2. G_3III_2

Метрика имеет вид $g_{13} = 1$, $g_{22} = V_2e^{-2x^3}$.
В ходе доказательства Теоремы 2 получили, что $\lambda = const$. Откуда следует, что $H \subset Hom(G_3III_3)$.

10.3. G_3III_3

Метрика имеет вид $g_{13} = e^{-x^3}$, $g_{22} = V_2$.
Выпишем уравнение Киллинга для пары пары (ik) равной (22)

$$V_2 b_2^2 = \frac{\lambda}{2}V_2.$$

Получаем, что $\lambda = const$ и $H \subset Hom(G_3III_3)$.

Вывод: для группы Ли G_3III класса 2 $H \subset Hom(G_3III)$.

11. G_3IV

11.1. G_3IV_1

Метрика имеет вид $g_{11} = V_1{x^3}^2e^{-2x^3}$, $g_{12} = -V_2x^3e^{-2x^3}$, $g_{13} = G_{13}e^{-x^3}$,
$g_{22} = V_2e^{-2x^3}$.
Выпишем уравнения Киллинга

$$V_2(b_i^3x^i + \beta^3)(x^3 - {x^3}^2) + V_2{x^3}^2b_1^1 - V_2x^3b_1^2 + G_{13}e^{x^3}b_1^3 = \frac{\lambda}{2}V_2{x^3}^2,$$
$$-(b_i^3x^i + \beta^3)V_2(1 - 2x^3) + V_2{x^3}^2b_2^1 - V_2x^3(b_1^1 + b_2^2) + V_2b_1^2 +$$
$$+G_{13}e^{x^3}b_2^3 = -\lambda V_2x^3,$$
$$-(b_i^3x^i + \beta^3)G_{13} + V_2{x^3}^2e^{-x^3}b_3^1 - V_2x^3e^{-x^3}b_3^2 + +G_{13}(b_1^1 + b_3^3) = \lambda G_{13},$$
$$-(b_i^3x^i + \beta^3)V_2 - V_2x^3b_3^1 + V_2b_2^2 = \frac{\lambda}{2}V_2,$$
$$-V_2x^3e^{-2x^3}b_3^1 + V_2e^{-2x^3}b_3^2 + G_{13}e^{-x^3}b_2^1 = 0,$$
$$2G_{13}e^{-x^3}b_3^1 = 0.$$

Получаем, что $\lambda = 0$ и $H \subset Isom(G_3IV_1)$.

11.2. G_3IV_2

Метрика имеет вид $g_{13} = G_{13}e^{-x^3}$, $g_{22} = -V_2e^{-2x^3}$, $g_{23} = -G_{13}x^3e^{-x^3}$.
Повторяя аналогичные вычисления, получаем $H \subset Isom(G_3IV_2)$.

Вывод: для группы Ли G_3IV класса 2 $H \subset Isom(G_3IV)$.

12. G_3V

В ходе доказательства Теоремы 2 получили $\lambda = const$. Следовательно, $H \subset Hom(G_3V)$.

13. G_3VI

13.1. G_3VI_1

Метрика имеет вид $g_{11} = V_2(e^{-x^3} - e^{-\alpha x^3})$, $g_{12} = e^{-(1+\alpha)x^3} - e^{-2\alpha x^3}$, $g_{13} = G_{13}e^{-x^3}$, $g_{22} = V_2e^{-2\alpha x^3}$.
Уравнения Киллинга имеют вид:

$$-(b_i^3x^i + \beta^3)V_2(e^{-x^3} - e^{-\alpha x^3})(e^{-x^3} - \alpha e^{-\alpha x^3}) + V_2(e^{-x^3} - e^{-\alpha x^3})b_1^1 +$$

$$+(e^{-(1+\alpha)x^3} - e^{-2\alpha x^3})b_1^2 + G_{13}e^{-x^3}b_1^3 = \frac{\lambda}{2}V_2(e^{-x^3} - e^{-\alpha x^3})^2,$$

$$-(b_i^3x^i + \beta^3)((1+\alpha)e^{-(1+\alpha)x^3} - 2\alpha e^{-2\alpha x^3}) + V_2(e^{-x^3} - e^{-\alpha x^3})^2b_2^1 +$$

$$+(e^{-(1+\alpha)x^3} - e^{-2\alpha x^3})(b_1^1 + b_2^2) + G_{13}e^{-x^3}b_2^3 + V_2e^{-2\alpha x^3}b_1^2 = \lambda(e^{-(1+\alpha)x^3} - e^{-2\alpha x^3}),$$

$$-(b_i^3x^i + \beta^3)G_{13}e^{-x^3} + V_2(e^{-x^3} - e^{-\alpha x^3})^2b_3^1 + (e^{-(1+\alpha)x^3} - e^{-2\alpha x^3})b_3^2 +$$

$$+G_{13}e^{-x^3}(b_1^1 + b_3^3) = \lambda G_{13}e^{-x^3},$$

$$-(b_i^3x^i + \beta^3)\alpha V_2e^{-2x^3} + (e^{-(1+\alpha)x^3} - e^{-2\alpha x^3})b_2^1 + V_2e^{-2\alpha x^3}b_2^2 = \frac{\lambda}{2}V_2e^{-2\alpha x^3},$$

$$(e^{-(1+\alpha)x^3} - e^{-2\alpha x^3})b_3^1 + V_2e^{-2\alpha x^3}b_3^2 + G_{13}e^{-x^3}b_2^1 = 0,$$

$$2G_{13}e^{-x^3}b_3^1 = 0.$$

Решая получившиеся уравнения, находим, что $\lambda = const$.
Тогда $H \subset Hom(G_3VI_1)$.

13.2. G_3VI_2

Метрика имеет вид $g_{13} = G_{13}e^{-\alpha x^3}$, $g_{22} = V_2e^{-2x^3}$.
Повторяя аналогичные вычисления, получаем $H \subset Hom(G_3VI_2)$.

13.3. G_3VI_3

Метрика имеет вид $g_{13} = G_{13}e^{-x^3}$, $g_{22} = V_2e^{-2\alpha x^3}$.
Выпишем уравнения Киллинга:

$$2G_{13}e^{-x^3}b_3^1 = 0,$$
$$V_2x^3e^{-2\alpha x^3}b_1^2 + G_{13}e^{-x^3}b_2^3 = 0,$$
$$-(b_i^3x^i + \beta^3)G_{13}e^{-x^3} + G_{13}e^{-x^3}(b_1^1 + b_3^3) = \lambda G_{13}e^{-x^3},$$
$$-\alpha(b_i^3x^i + \beta^3)V_2e^{-2\alpha x^3} + V_2e^{-2\alpha x^3}b_2^2 = \frac{\lambda}{2}V_2e^{-2\alpha x^3},$$
$$V_2e^{-2\alpha x^3}b_3^2 + G_{13}e^{-x^3}b_2^1 = 0,$$
$$2G_{13}b_3^1 = 0.$$

Уравнения имеют решения в двух случаях, если $b_i^3 = 0$ или $\alpha = \frac{1}{2}$.
В первом очевидно, что $\lambda = const$ и $H \subset Hom(G_3VI_1)$.
Во втором случае получаем $\lambda = \lambda(x)$, где $\lambda(x)$ - некоторая функция x, следовательно, $H \subset Conf(G_3VI_3)$.

14. G_3VI_0

14.1. $G_3VI_{0_1}$

Метрика имеет вид $g_{11} = V_2(e^{-x^3} - e^{x^3})^2$, $g_{12} = 1 - e^{2x^3}$, $g_{13} = G_{13}e^{-x^3}$, $g_{22} = V_2e^{2x^3}$.

14.2. $G_3VI_{0_2}$

Метрика имеет вид $g_{13} = G_{13}e^{x^3}$, $g_{22} = V_2e^{-2x^3}$.

14.3. $G_3VI_{0_3}$

Метрика имеет вид $g_{13} = G_{13}e^{-x^3}$, $g_{22} = V_2e^{2x^3}$.
Повторяя аналогичные вычисления, получаем $H \subset Hom(G_3VI_0)$.

15. G_3VII

Метрика имеет вид $g_{11} = \frac{1}{2}V_2e^{-2\alpha x^3}(1 - \cos 2\beta x^3)$, $g_{12} = -\frac{1}{2}V_2e^{-2\alpha x^3}\sin 2\beta x^3$, $g_{13} = G_{13}e^{-\alpha x^3}\cos\beta x^3$, $g_{22} = \frac{1}{2}V_2e^{-2x^3}(1 + \cos 2\beta x^3)$, $g_{23} = G_{13}e^{-\alpha x^3}\sin\beta x^3$.

16. G_3VII_0

Метрика имеет вид $g_{11} = \frac{1}{2}V_2(1-\cos 2x^3)$, $g_{12} = -\frac{1}{2}V_2 \sin 2x^3$, $g_{13} = G_{13}\cos x^3$, $g_{22} = \frac{1}{2}V_2(1+\cos 2x^3)$, $g_{23} = G_{13}\sin x^3$.

Повторяя аналогичные вычисления, получаем $H \subset Hom(G_3VII_0)$.

Теорема 3 доказана.

Литература

1. Абдрахимова, Н.Р. Классификация аффинных порядков на трехмерных связных односвязных разрешимых группах Ли / Н.Р. Абдрахимова. // IX Всесоюзная геометрическая конференция. Тезисы докладов. – Кишинев. – 1988. – С. 3.
2. Абдрахимова, Н.Р. Синтетическая теория аффинных лоренцевых многообразий и упорядоченных групп Ли / Н.Р. Абдрахимова, А.К. Гуц, Н.Л. Шаламова // Докл. АН СССР. – 1988. – Т. 303, N. 4. – С. 777–781.
3. Александров, А.Д. Конусы с транзитивной группой / А.Д. Александров // Докл. АН СССР. – 1969. – Т. 189, N. 4. – С. 695–698.
4. Винберг, Э.Б. Теория однородных выпуклых конусов / Э.Б. Винберг // Тр. ММО. – 1963. – Т. 12. – С. 302–358.
5. Гаврилов, С.П. Левоинвариантные метрики на однородных разрешимых группах Ли./ С.П. Гаврилов // Теория относительности и гравитация. – Казань: КГУ. – 1985. – N. 22. – С. 31–64.
6. Гуц, А.К. Порядковые и пространственно-временные структуры на однородных многообразиях / А.К. Гуц // Дис. док. физ.-мат. наук. – Новосибирск: ИМ СО АН СССР – 1987. – 204 с.
7. Yamagychi, S. On complete affinely flat structures of some solvable Lie groups / S. Yamagychi // Mem. of Faculty of Scince Kyuchu Univ. – 1979. – V. A33. – P. 209–218.
8. Milnor, J. On fundamental groups of complete affinely flat manifolds / J. Milnor // Advances in Math. – 1977. – V. 25., N. 2. – P. 178-187.

Математические структуры и моделирование
2009, вып. 20, с. 48–51

УДК 514.12

КОНФОРМНЫЕ ПРЕОБРАЗОВАНИЯ ЛОРЕНЦЕВОЙ МЕТРИКИ

А.Н. Романов

В статье рассматриваются лоренцевы многообразия, которые, в отличие от римановых, имеют метрики сигнатуры $(-\,+\,+ \ldots +)$. Приложения лоренцевой геометрии, рассматривающей лоренцевы многообразия, используются в разделах теоретической физики, изучающих свойства пространства-времени.

В статье будут рассматриваться лоренцевы многообразия, в которых условие конечности лоренцевой функции расстояния является инвариантом относительно конформных преобразований метрики (терминология взята из [1]).

В дальнейшем будем использовать следующее утверждение (см. [1], теорема 3.30):

Лемма 1. *Пространство-время (M, g) глобально гиперболично тогда и только тогда, когда оно сильно причинно и (M, g') удовлетворяет условию конечности расстояния для всех $g' \in C(M, g)$.*

Здесь через $C(M, g)$ обозначен класс лоренцевых метрик на многообразии M, глобально конформных метрике $g : g' \in C(M, g) \Leftrightarrow g' = \Omega g$ для некоторой гладкой функции $\Omega : M \to (0, \infty)$.

Разделим лоренцевы многообразия на два непересекающихся класса: A и B. Класс B характеризуется следующим свойством. Пусть между точками $p, s \in M$ выполнены следующие соотношения: $s \in cl(J_p^+)$, но $s \notin J_p^+$. Таким образом любую окрестность U_s точки $s \in M$ можно достичь направленной в будущее причинной кривой γ, выходящей из p, однако сама точка s остаётся недостижимой. Допустим теперь, что имеет место следующая ситуация: существует настолько малая окрестность U_s точки s, что для того, чтобы достичь её направленной в будущее причинной кривой, выходящей из p, необходимо, чтобы, во-первых, эта кривая γ целиком находилась бы в некотором (фиксриманованном) компактном множестве K, а, во-вторых, её риманова длина (измеренная в любой заранее выбранной римановой метрике) была бы больше любого заранее заданного положительного числа N. Другими словами, чтобы «подойти» достаточно

Copyright © 2009 **А.Н. Романов.**
Омский государственный университет.
E-mail: aroms@bk.ru

близко к точке s, причинная кривая γ должна совершить достаточно большое количество «оборотов» во множестве K.

Если такая ситуация имеет место в некотором многообразии (M, g), то будем относить его к классу B, в противном случае будем считать данное лоренцево многообразие относящимся к классу A.

В качестве гипотезы можно выдвинуть предположение, что к классу B относятся лишь многообразия, не являющиеся причинными, то есть содержащие замкнутые причинные кривые. Однако это утверждение требует отдельного доказательства. В дальнейшем же будут приведены результаты, касающиеся только лоренцевых многообразий, принадлежащих классу A (даже если это не будет отдельно оговорено).

Следующее утверждение позволяет делать некоторые выводы о топологии лоренцева многообразия, исходя из его причинной структуры.

Лемма 2. *Пусть пространство-время (M, g) принадлежит классу A. Если для некоторых точек $p, s \in M$ множество $J_p^+ \cap J_s^-$ не замкнуто в M, а $I_p^+ \cap I_s^- \neq \emptyset$, то тогда (замкнутое) множество $cl(J_p^+ \cap J_s^-)$ не является компактным.*

Доказательство. Допустим, что множество $cl(J_p^+ \cap J_s^-)$ компактно. Так как множество $J_p^+ \cap J_s^-$ не замкнуто, то существует точка $q \in cl(J_p^+ \cap J_s^-)$ такая, что $q \notin J_p^+ \cap J_s^-$. В этом случае $q \notin J_p^+$ (случай $q \notin J_s^-$ доказывается аналогично).

Рассмотрим последовательность точек $q_n \subset J_p^+ \cap J_s^-$ такую, что $q_n \to q$, т.е. сходящуюся к q (сходимость в исходной топологии многообразия M).

Таким образом, имеем:

$$q_n : p \leq q_n, q_n \to q.$$

Так как $p \leq q_n$, то $\forall n$ существует причинная кривая γ_n, идущая из p в q_n. Продолжим γ_n до непродолжаемой причинной кривой. Любая окрестность точки q содержит все точки q_n, начиная с некоторого n. А так как $q_n \in \gamma_n$, то q является точкой накопления последовательности причинных непродолжаемых кривых γ_n. Отсюда следует (см. [1], предложение 2.18), что существует причинная непродолжаемая кривая γ, являющаяся предельной для последовательности γ_n, и такая, что $q \in \gamma$. Выберем параметризацию γ так, что $\gamma : (-\infty, \infty) \to M$ и $\gamma(0) = q$, причем уменьшение параметра t кривой γ соответствует движению по ней в прошлое.

Рассмотрим часть кривой γ, идущую в прошлое от точки q : $\gamma(-\infty, 0]$. Заметим, что для любой точки $a \in \gamma(-\infty, 0]$ выполняется соотношение: $a \in cl(J_p^+)$. Действительно, т.к. γ – предельная кривая последовательности γ_n, то существует подпоследовательность $\gamma_m \subset \gamma_n$ такая, что для любой точки $a \in \gamma$ каждая ее окрестность U_a пересекает все, за исключением конечного числа, кривые из γ_m. Взяв точки r_m т.ч.: $r_m \in \gamma_m, r_m \in U_a$, получим сходящуюся к a последовательность $r_m : r_m \to a$. Если выполнено еще соотношение $r_m \in J_p^+$, то получим, что $a \in cl(J_p^+)$. В данном случае включение $r_m \in J_p^+$ выполняется всегда. В самом деле, если $r_m \notin J_p^+$, то это означает, что кривая γ (вместе с кривыми γ_m)

покинула область $cl(J_p^+)$. Однако выйти из $cl(J_p^+)$ γ может лишь через точку p, так как все γ_m «фокусируются» в p (по их определению), а γ – предельная кривая для последовательности γ_m. Но такого быть не может, так как это означало бы существование отрезка (лежащего на кривой γ), соединяющего точки p и q и являющегося частью причинной кривой (γ – причинна), что противоречит выбору точки $q: q \notin J_p^+$.

Таким образом, мы показали, что $\forall a \in \gamma(-\infty, 0]$, $a \in cl(J_p^+)$. Ясно, что выполнено также включение $a \in cl(J_p^+ \cap J_s^-)$ (т.к. из $a \leq q, q << r \Rightarrow , a << r$, т.е. $a \in int\, J_s^-$). В результате имеем: часть кривой γ, идущая в прошлое от точки q, целиком находится во множестве $cl(J_p^+ \cap J_s^-)$, которое по сделанному предположению является компактным.

По построению кривой γ (см. [1], предложение 2.18), последовательность γ_m сходится к γ равномерно на любом компактном множестве из $\mathbf{R}$ в случае, если кривые γ и γ_m параметризованы длиной дуги, вычисленной относительно (полной) римановой метрики.

Так как ни для какого значения параметра $t \leq 0$ кривая γ не покидает множества $cl(J_p^+ \cap J_s^-)$, а последовательность γ_m сходится к γ равномерно на любом компактном множестве из $\mathbf{R}$ (то есть кривые γ_m «повторяют» движение γ), то получаем следующую ситуацию: если взять достаточно малую окрестность U_q точки q, то длины кривых γ_m, достигающих этой окрестности, с необходимостью должны быть больше любого наперёд заданного положительного числа N. Однако это означает, что пространство-время (M, g) принадлежит классу B, в то время как по условию (M, g) принадлежит классу A.

Полученное противоречие опровергает сделанное предположение о том, что множество $cl(J_p^+ \cap J_s^-)$ компактно и тем самым доказывает лемму. ■

Следующее утверждение взято из [1].

Лемма 3. *Пусть (M, g) – пространство-время. Если для некоторых точек $p, s \in M$ множество $cl(J_p^+ \cap J_s^-)$ не является компактным, то существует лоренцева метрика $g' \in C(M, g)$, глобально конформная метрике g, такая, что $d(a, b) = \infty$ для некоторых точек $a, b \in M$.*

Теперь применим полученные результаты к исследованию причинной структуры пространства-времени (M, g), для которого условие конечности расстояния является инвариантом при конформных преобразованиях метрики g.

Теорема 1. *Пусть (M, g) – различающее пространство-время. Если пространство-время (M, g') удовлетворяет условию конечности расстояния для всех $g' \in C(M, g)$, то пространство-время (M, g) является глобально гиперболическим.*

Доказательство. Покажем сначала, что (M, g) является причинно простым, (т.е. различающим с дополнительным условием, что множества J_p^+ и J_p^- замкнуты для всех $p \in M$).

Покажем, что множество J_p^+ замкнуто для любой точки $p \in M$ (замкнутость J_p^- доказывается аналогично).

Допустим обратное: $\exists$ точка $q \in cl(J_p^+) \setminus J_p^+$. Возьмем в I_q^+ произвольную точку r. Покажем, что множество $J_p^+ \cap J_r^-$ не пусто. Так как $q \in cl(J_p^+)$, то $\exists$ последовательность точек $q_n \subset J_p^+$, сходящаяся к q (сходимость в исходной топологии многообразия M). Так как $q \in I_r^-$ а множество I_r^- открыто (см [1], лемма 2.5), то для достаточно больших $n\, q_n \in I_r^-$, т.е.$q_n << r$. Тогда из соотношений $p \leq q_n, q_n << r$ получаем: $p << q_n$ т.е. $r \in I_p^+$. Таким образом, имеем: множество $I_p^+ \cap I_r^-$ не пусто.

Получаем: $J_p^+ \cap J_r^- \neq \emptyset$ (т.к. $I_p^+ \subset J_p^+, I_r^- \subset J_r^-, I_p^+ \cap I_r^-$).

Множество $J_p^+ \cap J_r^-$ не замкнуто в M $(q \in I_r^- = intJ_r^-)$:

$$(q \in I_r^- = intJ_r^-) : q \in int(J_r^-)\,,\, q \in cl(J_p^+),\, q \notin J_p^+ \;\Rightarrow\; q \in cl(J_p^+ \cap J_r^-) \setminus (J_p^+ \cap J_r^-).$$

Тогда по лемме 2 получаем, что множество $cl(J_p^+ \cap J_r^-)$ некомпактно. Следуя лемме 3, можно найти метрику $g' \in C(M, g)$ такую, что пространство-время (M, g') не удовлетворяет условию конечности лоренцева расстояния, что противоречит условию теоремы.

Таким образом, множества J_p^+ и J_p^- замкнуты и пространство-время (M, g) является причинно простым. Причинная простота пространства-времени (M, g) автоматически влечёт за собой его сильную причинность. Теперь, по лемме 1, сразу получаем желаемый результат: пространство-время (M, g) является глобально гиперболическим. ■

Литература

1. Бим, Дж. Глобальная лоренцева геометрия / Дж. Бим, П. Эрлих. – М.: Мир, 1985.
2. Пенроуз, Р. Структура пространства-времени / Р. Пенроуз. – М.: Мир, 1972.
3. Malament, D.B. The class of continuous timelike curves determines the topology of spasetime / D.B. Malament // J. Math. Phys. – 1977. – V. 18, N. 7. – P. 1399-1404.

Математические структуры и моделирование
2009, вып. 20, с. 52–73

УДК 004.92,532.5,519.633

МОДЕЛИРОВАНИЕ ДИНАМИКИ ФЛЮИДОВ НА ГРАФИЧЕСКОМ ПРОЦЕССОРЕ

Ю.В. Фролова, В.В. Коробицын

An approach to fluid simulation on the GPU is described. The algorithm is based on the physical equations of fluid flow, namely the Navie–Stokes equations. We provide clear explanations and draw connections between the math and its implementation. It describes the techniques to perform the simulation on the GPU.

1. Введение

Флюиды (т.е. жидкости и газы) присутствуют везде: вода в реке, текущая вдоль берега, струйка дыма от сигареты, клубы пара из кипящего чайника, испарения, формирующиеся в облака, смешиваемые краски в банке. В основе всего этого — поток флюидов. Всё это явления, которые многие хотели бы изобразить реалистично в интерактивных графических приложениях. Рисунок 1 показывает примеры смоделированных флюидов.

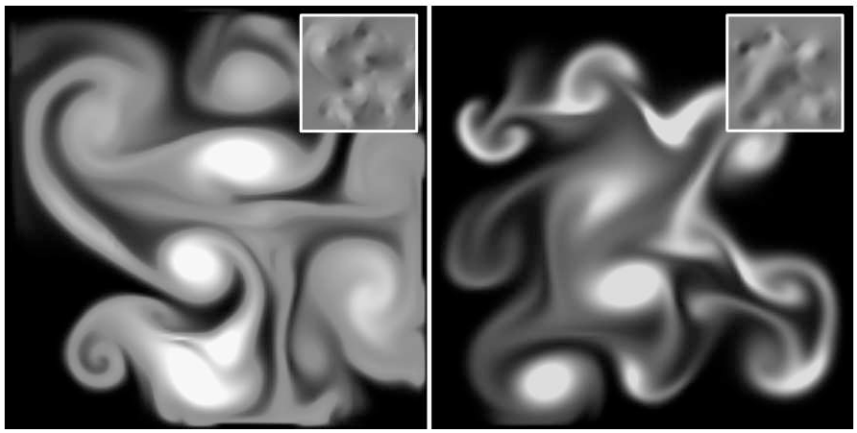

Рис. 1. Имитация вихревого движения краски. В маленькой картинке представлена текстура поля скорости

Copyright © 2009 **Ю.В. Фролова, В.В. Коробицын.**
Омский государственный университет.
E-mail: juliavfrolova@rambler.ru

Моделирование флюидов является полезным кирпичиком, лежащим в основе имитации различных природных явлений. Из-за большого количества параллелизма в графических аппаратных средствах прогоны моделирования значительно быстрее выполняются на графическом процессоре (GPU), чем на центральном процессоре (CPU). Используя современные видеокарты NVIDIA GeForce, можно достигнуть ускорения в несколько раз над эквивалентным моделированием на CPU.

Цель представленной работы состоит в раскрытии возможностей и помощи в изучении мощного средства. Без понимания основ физики и математики флюидов использование и развитие алгоритмов не представляется возможным. По этой причине в работе подробно рассмотрена математическая часть, содержащая много уравнений. По возможности даются ясные объяснения и показываются связи между математическими формулами и компьютерной реализацией. Для написания данной статьи были использованы работы М. Харриса [5–8], Дж. Крюгера и Р. Вестерманна [9], В. Ли, Ж. Фана, К. Вейя и А. Кауфмана [10], Дж. Стама [11, 12].

Предполагается, что читатель знаком с курсами дифференциального и интегрального исчисления, дифференциальных уравнений и векторного исчисления. Будет полезен опыт с конечно-разностным приближением производных.

Описываемая методика основана на методе «устойчивых флюидов» [11]. Однако Дж. Стам моделировал, используя реализацию на CPU, здесь же будет описана реализация на графических аппаратных средствах. GPU хорошо подходит для вычислений, требующихся для моделирования флюидов. Представляемая модель выполнена на *сетке ячеек*. Программируемые GPU оптимизированы для выполнения вычислений над пикселями, которые можно рассматривать как ячейки сетки. GPU достигает высокой скорости вычислений за счет параллелизма: способность обрабатывать огромное число вершин и пикселей одновременно. Также оптимизировано выполнение многократного поиска по текстуре за один цикл. Поскольку имитационная сетка будет помещаться на текстурах, то скорость и параллелизм — это то, что нам нужно.

Эта статья не раскрывает все вопросы гидродинамики. Ограничимся моделированием только непрерывного объема флюида в двумерной прямоугольной области. Также не имитируются свободные границы поверхности между флюидами, как, например, граница раздела между всплесками воды и воздуха.

Во всей работе используются единые математические обозначения. В уравнениях *наклонный* шрифт используется для обозначения переменных, которые представляют скалярные величины, например давление p. **Жирный** шрифт используется для представления векторных величин, например скорость $\mathbf{u}$. Все векторы в этой статье двумерны.

В пункте 2 описывается математическая основа метода, включая рассмотрение уравнений, которые управляют потоком флюида. Затем обсуждается подход к решению уравнений. Пункт 3 описывает реализацию моделирования флюида на GPU. Пункт 4 представляет некоторые примеры моделирования флюидов.

2. Математическая основа

Для имитации поведения флюида необходимо иметь математическое описание состояния флюида в каждый момент времени. Наиболее важным параметром для представления флюида является его скорость, поскольку скорость определяет как перемещение самого флюида, так и предметов, которые в нем находятся. Скорость флюида изменяется как во времени, так и в пространстве, поэтому представляется как векторное поле.

Векторное поле есть некоторое отображение векторнозначной функции в параметризованное пространство, например декартову сетку. (Другие параметризации пространства также возможны, но мы принимаем двумерную декартову сетку.) Векторное поле скорости флюида определено из условия, что для каждой позиции $\mathbf{x} = (x, y)$, есть сопоставленная скорость, изменяющаяся во времени t, $\mathbf{u}(\mathbf{x}, t) = (u(\mathbf{x}, t), v(\mathbf{x}, t))$, как показано на рисунке 2.

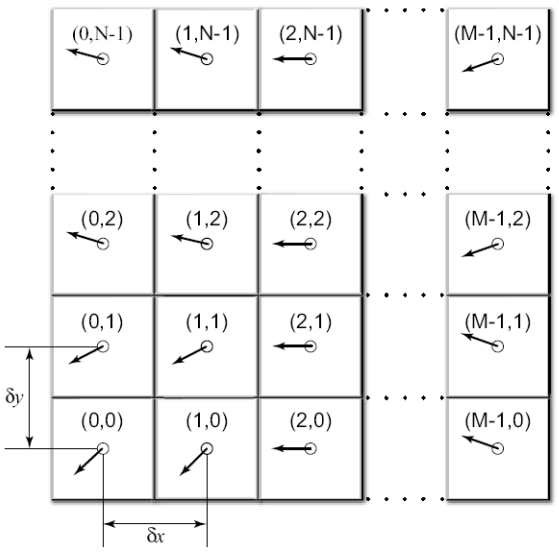

Рис. 2. Векторная сетка флюида

Ключевым для моделирования флюида являются шаги во времени, на каждом временном шаге нужно правильно определить текущее поле скорости. Это можно сделать решая уравнения, которые описывают изменения поля скоростей со временем под действием ряда сил. Как только у нас будет поле скоростей, мы можем сделать интересные вещи с ним, например, используем его для перемещения объектов или для отображения плотности дыма.

2.1. Уравнения движения Навье-Стокса

Флюид является несжимаемым, если масса любой его части постоянна во времени. Флюид однороден, если его плотность ρ не изменяется в пространстве. Комбинация несжимаемости и однородности означает, что плотность является константой как с течением времени, так и в пространстве. Эти предположения

являются общими в гидродинамике и применяются для моделирования реальных флюидов, например воды и воздуха.

Рассматриваемая модель флюида реализуется на регулярной декартовой сетке с пространственными координатами $\mathbf{x} = (x, y)$ и переменной времени t. Флюид представлен полем скоростей $\mathbf{u}(\mathbf{x}, t)$ и скалярным полем давлений $p(\mathbf{x}, t)$. Эти поля изменяются как в пространстве, так и во времени. Если скорость и давление известны в начальный момент времени $t = 0$, тогда состояние флюида во времени может быть описано уравнениями Навье-Стокса для несжимаемого потока:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{F}, \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{2}$$

где ρ — (константа) плотность флюида, ν — кинематическая вязкость и вектор $\mathbf{F} = (f_x, f_y)$ представляет любые внешние силы, которые действуют на флюид. Заметим, что уравнение (1) — это два уравнения, поскольку $\mathbf{u}$ — векторная величина:

$$\frac{\partial u}{\partial t} = -(\mathbf{u} \cdot \nabla)u - \frac{1}{\rho}\nabla p + \nu \nabla^2 u + f_x,$$

$$\frac{\partial v}{\partial t} = -(\mathbf{u} \cdot \nabla)v - \frac{1}{\rho}\nabla p + \nu \nabla^2 v + f_y.$$

Таким образом, имеем три неизвестных u, v и p и три уравнения.

Уравнения Навье-Стокса легче понять, разбирая их на простые части. Сначала попытаемся понять члены, влияющие на поток флюида, их четыре в правой части уравнения (1). Рассмотрим каждый в отдельности.

2.2. Члены в уравнениях Навье-Стокса

Перемещение потока. Скорость флюида способствует перемещению объектов, плотностей и других величин вместе с потоком. Представим попадание струи краски в движущуюся жидкость. Краска перемещается вдоль поля скорости флюида. Фактически скорость флюида несет флюид параллельно, подобно тому как она несет краску. Первый член из правой части уравнения (1) представляет собой само-адвекцию поля скоростей и назван *адвекцией* (перемещением потока).

Давление. Поскольку молекулы флюида могут перемещаться вокруг друг друга, они стремятся к «всплескам» и «хлюпам». Когда некоторая сила приложена к флюиду, она немедленно распространяется через весь объем. Молекулы, близкие к месту приложения силы, наталкиваются на те, которые дальше от него, и создается давление. Поскольку давление есть сила на единицу области, любое давление во флюиде естественно ведет к ускорению. (Вспомним второй закон Ньютона, $F = ma$.) Второй член уравнения, названный *давлением*, представляет ускорение флюида.

Диффузия. Из опыта с реальными флюидами известно, что некоторые жидкости «плотнее» других. Например, сахарный сироп струится медленнее, а этиловый спирт быстрее воды. Говорят, что более плотные флюиды имеют более высокую *вязкость*. Вязкость является мерой того, как жидкость сопротивляется течению. Это сопротивление заканчивается диффузией движущей силы (следовательно, и скорости), так что третий член называем *диффузией*.

Внешние силы. Четвертый член представляет ускорение, возникающее в результате действия внешних сил, приложенных к флюиду. Эти силы могут быть либо *локальными*, либо *объектными* силами. Локальные силы приложены к отдельной области флюида, например сила вентилятора, создающего поток воздуха. Объектная сила прикладывается ко всему флюиду, например сила тяжести.

2.3. Краткий обзор векторного исчисления

Уравнения (1) и (2) содержат символ ∇, известный как набла-оператор. Три способа применения набла-оператора — градиент, дивергенция и оператор Лапласа – приведены в таблице 1. В столбце «символ» приводятся обозначения операторов, принятые в отечественной литературе. Индексы i и j, использованные в выражениях в таблице, указывают на дискретные позиции на декартовой сетке, δx и δy — шаги сетки по осям x и y соответственно.

Таблица 1. Операторы векторного исчисления, используемые в гидродинамике

оператор	определение	символ	конечно-разностная форма
Градиент	$\nabla p = \left(\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y}\right)$	grad p	$\frac{p_{i+1,j}-p_{i-1,j}}{2\delta x}, \frac{p_{i,j+1}-p_{i,j-1}}{2\delta y}$
Дивергенция	$\nabla \cdot \mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$	div $\mathbf{u}$	$\frac{u_{i+1,j}-u_{i-1,j}}{2\delta x} + \frac{v_{i,j+1}-v_{i,j-1}}{2\delta y}$
Лапласа	$\nabla^2 p = \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2}$	$\triangle p$	$\frac{p_{i+1,j}-2p_{i,j}+p_{i-1,j}}{(\delta x)^2} + \frac{p_{i,j+1}-2p_{i,j}+p_{i,j-1}}{(\delta y)^2}$

Градиент скалярного поля есть вектор частных производных скалярного поля. Дивергенция, которая появляется в уравнении (2), имеет важное физическое значение. Это интенсивность источника потока из данной точки пространства. В уравнениях Навье-Стокса оператор дивергенции приложен к скорости потока и показывает изменение скорости через поверхность, окружающую небольшую часть флюида. Уравнение неразрывности (2) предполагает несжимаемость флюида, то есть флюид должен иметь нулевую дивергенцию. Скалярное произведение в операторе дивергенции дает в результате сумму частных производных, а не вектор, как оператор градиента. Это означает, что оператор дивергенции может прилагаться только к векторному полю, например скорости $\mathbf{u} = (u, v)$.

Отметим, что градиент скалярного поля — векторное поле, а дивергенция векторного поля является скалярным полем. Если оператор дивергенции при-

ложен к результату оператора градиента, то результатом является оператор Лапласа $\nabla \cdot \nabla = \nabla^2$. Для упрощения примем, что ячейки сетки квадратные ($\delta x = \delta y$), тогда оператор Лапласа имеет конечно-разностную форму

$$\nabla^2 p = \frac{p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1} - 4p_{i,j}}{(\delta x)^2}. \tag{3}$$

Оператор Лапласа встречается в физике в форме уравнений диффузии, например уравнение теплопроводности. Уравнение вида $\nabla^2 x = b$ известно как уравнение Пуассона. Когда $b = 0$ — уравнение Лапласа. В уравнении (1) оператор Лапласа приложен к векторному полю. Это символьное упрощение: оператор прилагается отдельно к каждому скалярному компоненту векторного поля.

2.4. Решение уравнения Навье-Стокса

Уравнения Навье-Стокса можно решить аналитически только для некоторых простых физических форм. Тем не менее, можно использовать методы численного интегрирования для пошагового решения. Поскольку нас интересует наблюдение динамики потока во времени, то пошаговое численное решение нам вполне подходит.

Как с любым алгоритмом, решение уравнений Навье-Стокса необходимо разбить на простые шаги. Используемый нами метод основан на методике устойчивых флюидов, описанной в [11]. Здесь приводится описание каждого шага, а в пункте 3 описывается их реализация с использованием языка Cg для GPU.

Сначала необходимо преобразовать уравнение к виду, приемлемому для численного решения. Вспомним, что уравнения Навье-Стокса — это три скалярных уравнения, которые необходимо решить для поиска величин u, v и p. Тем не менее, поиск решения не очевиден. Далее описывается преобразование, ведущее к простому алгоритму.

Разложение Гельмгольца-Ходжа. Любой вектор $\mathbf{v}$ может быть разложен по базисным векторам. Например, векторы на декартовой сетке представляются как пара отрезков вдоль осей сетки: $\mathbf{v} = (x, y)$. Тот же вектор может быть записан: $\mathbf{v} = x\mathbf{i} + y\mathbf{j}$, где $\mathbf{i}$ и $\mathbf{j}$ — базисные вектора, направленные вдоль осей сетки.

Так же можно разложить вектор на сумму векторов, а поле векторов — на сумму векторных полей. Пусть D — область в пространстве или в нашем случае на плоскости, в которой определен наш флюид. Пусть данная область имеет гладкую границу ∂D (то есть дифференцируемую) с вектором нормали n. Можно использовать теорему Гельмгольца–Ходжа о разложении вектора, приведенную в [2].

Теорема о разложении Гельмгольца-Ходжа. *Векторное поле* $\mathbf{w}$ *на* D *можно однозначно разложить на составные части вида:*

$$\mathbf{w} = \mathbf{u} + \nabla p, \tag{4}$$

где $\mathbf{u}$ *имеет нулевую дивергенцию и параллельно к* ∂D*, то есть* $\mathbf{u} \cdot \mathbf{n} = 0$ *на* ∂D*.*

Теорема приводится без доказательства. Подробнее смотрите [2, с. 37–38].

Эта теорема указывает, что любое векторное поле может быть разложено на сумму двух других векторных полей: векторное поле с нулевой дивергенцией и градиент скалярного поля. При этом поле с нулевой дивергенцией стремится к нулю на границе. Это мощное средство, ведущее нас к двум полезным реализациям.

Первая реализация. Численное решение уравнений Навье-Стокса разделяется на три этапа вычисления скорости на каждом временном шаге: адвекция, диффузия и приложение сил. Результатом является новое поле скорости $\mathbf{w}$ с ненулевой дивергенцией. Но уравнение неразрывности требует, чтобы мы завершали временной шаг со скоростью с нулевой дивергенцией. К счастью, теорема о декомпозиции Гельмгольца-Ходжа утверждает, что дивергенция скорости может быть скорректирована вычитанием градиента получившегося поля давления

$$\mathbf{u} = \mathbf{w} - \nabla p. \tag{5}$$

Вторая реализация. Теорема также дает метод для вычисления поля давления. Если применить оператор дивергенции к обеим частям уравнения (4), то получим

$$\nabla \cdot \mathbf{w} = \nabla \cdot (\mathbf{u} + \nabla p) = \nabla \cdot \mathbf{u} + \nabla^2 p. \tag{6}$$

В силу уравнения (2), $\nabla \cdot \mathbf{u} = 0$, упростим

$$\nabla^2 p = \nabla \cdot \mathbf{w}, \tag{7}$$

а это есть уравнение Пуассона (см. п. 2.3) для давления флюида. Это означает, что после перехода к дивергенции скорости $\mathbf{w}$ можно решить уравнение (7) для p, затем использовать $\mathbf{w}$ и p для вычисления нового поля $\mathbf{u}$ с нулевой дивергенцией, используя уравнение (5). Обратимся к этому позже.

Теперь нам необходим способ для вычисления $\mathbf{w}$. Вернемся к сопоставлению векторов и векторных полей. Из определения скалярного произведения известно, что можно найти проекцию вектора $\mathbf{r}$ на единичный вектор $\mathbf{s}$, вычислив скалярное произведение $\mathbf{r}$ и $\mathbf{s}$. Скалярное произведение есть оператор проекции для векторов, который отображает вектор $\mathbf{r}$ на свою компоненту в направлении $\mathbf{s}$. Можно использовать теорему о декомпозиции Гельмгольца-Ходжа для определения оператора проекции $\mathbb{P}$, который проектирует векторное поле $\mathbf{w}$ на векторное поле $\mathbf{u}$ с нулевой дивергенцией. Если применить оператор $\mathbb{P}$ к уравнению (4), получим

$$\mathbb{P}\mathbf{w} = \mathbb{P}\mathbf{u} + \mathbb{P}(\nabla p).$$

Но по определению $\mathbb{P}$, $\mathbb{P}\mathbf{w} = \mathbb{P}\mathbf{u} = \mathbf{u}$. Следовательно, $\mathbb{P}(\nabla p) = 0$. Теперь используем эти идеи для упрощения уравнения Навье-Стокса.

Сначала применим оператор проекции к обеим частям уравнения (1):

$$\mathbb{P}\frac{\partial \mathbf{u}}{\partial t} = \mathbb{P}\left(-(\mathbf{u}\cdot\nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + \mathbf{F}\right).$$

Поскольку $\mathbf{u}$ имеет нулевую дивергенцию, то производная в левой части уравнения $\mathbb{P}(\partial\mathbf{u}/\partial t) = \partial\mathbf{u}/\partial t$, а также $\mathbb{P}(\nabla p) = 0$, таким образом, член давления исчезает. Остается уравнение в следующем виде

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbb{P}\left(-(\mathbf{u}\cdot\nabla)\mathbf{u} + \nu\nabla^2\mathbf{u} + \mathbf{F}\right). \tag{8}$$

Важно, что это уравнение заключает в себе последовательность шагов алгоритма моделирования потока флюида. Сначала вычислим то, что находится в круглых скобках в правой части уравнения. Последовательно вычисляем адвекцию, диффузию и приложение сил. Это дает в результате дивергенцию поля скоростей $\mathbf{w}$, к которому применяем оператор проекции, чтобы получить новое поле $\mathbf{u}$ с нулевой дивергенцией. Для этого решаем уравнение (7) для p, затем вычитаем градиент p из $\mathbf{w}$, как в выражении (5).

Обычно при реализации, компоненты не вычисляются отдельно и не складываются, как в уравнении (8). Взамен этого решение находится как последовательность преобразований векторного поля. Другими словами, каждый компонент есть шаг, который берет поле на вход и производит новое поле на выход. Можно определить оператор $\mathbb{S}$, который эквивалентен решению уравнения (8) за один шаг по времени. Оператор определяется как последовательность операторов адвекции ($\mathbb{A}$), диффузии ($\mathbb{D}$), приложение сил ($\mathbb{F}$) и проекция ($\mathbb{P}$):

$$\mathbb{S} = \mathbb{P}\circ\mathbb{F}\circ\mathbb{D}\circ\mathbb{A}. \tag{9}$$

Таким образом, шаг алгоритма моделирования может быть выражен

$$\mathbb{S}(\mathbf{u}) = \mathbb{P}\circ\mathbb{F}\circ\mathbb{D}\circ\mathbb{A}(\mathbf{u}).$$

Операторы применяются справа налево; первый – адвекция, за которым следует диффузия, приложение сил и проекция. Заметим, что время опускается здесь для ясности, но на практике шаг по времени должен быть использован в вычислении каждого оператора. Теперь разберем более внимательно шаги адвекции и диффузии, а затем подойдем к решению уравнений Пуассона.

Перемещение потока. Адвекция — процесс, посредством которого скорость флюида перемещает флюид и другие величины во флюиде. Чтобы вычислить результат адвекции, необходимо скорректировать значения величины в каждой точке сетки. Поскольку мы вычисляем, как величина перемещается вдоль поля скорости, то легче представить каждую ячейку сетки как частицу. Сначала попытаемся вычислить результат адвекции, для этого обновим значения величин на сетке так, как будто двигается система частиц. Переместим позицию $\mathbf{r}$ каждой частицы вперед вдоль поля скоростей на расстояние, которое она пройдет за время dt:

$$\mathbf{r}(t+\delta t) = \mathbf{r}(t) + \mathbf{u}(t)\delta t.$$

Вы можете понимать это как метод Эйлера. Это простой метод для явного интегрирования обыкновенных дифференциальных уравнений. Имеются более точные методы, например метод средней точки или методы Рунге-Кутты.

Имеется две проблемы с этим подходом: первая это то, что явные методы неустойчивы при больших шагах по времени, и решение может уйти в бесконечность («взорваться»), если величина $\mathbf{u}(t)dt$ больше размера одной ячейки сетки. Вторая проблема характерна для реализаций на GPU. При моделировании с использованием фрагментных шейдеров невозможно изменить позиции фрагментов. А этот метод интегрирования требует возможности «перемещать» частицы, что не может быть реализовано на имеющихся GPU.

Для решения первой проблемы воспользуемся неявным методом [11]. Прежде чем вычислить адвекцию величины, переместим частицу из текущей позиции в предыдущую, используя текущую скорость и отрицательный шаг по времени. Вычисляем величину в этой точке и переносим ее в стартовую ячейку сетки. Чтобы обновить величину q (это может быть скорость, плотность, температура или любая величина, несомая флюидом), используем следующее уравнение:

$$q\Big(\mathbf{x}, t+\delta t\Big) = q\Big(\mathbf{x} - \mathbf{u}(\mathbf{x}, t)\delta t, t\Big). \tag{10}$$

Не только легко реализовать этот метод на GPU, но и, как показал Дж. Стам, решение будет устойчивым для произвольных шагов по времени и любых скоростей. На рисунке 3 представлено вычисление адвекции в ячейке, выделенной двойным кругом. Поле скорости перемещается назад во времени на вектор $\mathbf{x}$. С помощью билинейной интерполяции четырех величин сетки, соседних к $\mathbf{x}$ (помечены квадратом), получаем результат, который записывается в стартовую ячейку сетки.

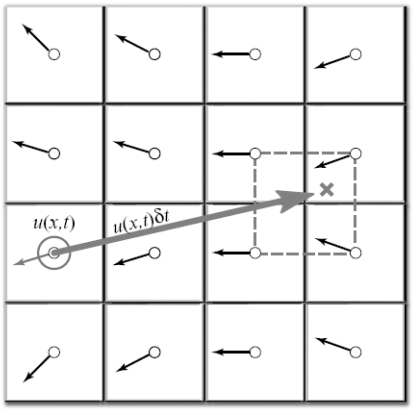

Рис. 3. Вычисление адвекции флюида

Диффузия вязкости. Как отмечалось ранее, вязкие флюиды сопротивляются перемещению в потоке, что является результатом диффузии (или рассеивания) скорости. Дифференциальное уравнение в частных производных для

диффузии вязкости

$$\frac{\partial \mathbf{u}}{\partial t} = \nu \nabla^2 \mathbf{u}. \tag{11}$$

Так же, как для адвекции, есть несколько методов решения этого уравнения. Простой подход состоит в использовании явной схемы

$$\mathbf{u}(\mathbf{x}, t + \delta t) = \mathbf{u}(\mathbf{x}, t) + \nu \delta t \nabla^2 \mathbf{u}(\mathbf{x}, t).$$

В этом уравнении ∇^2 — дискретная форма оператора Лапласа (3). Подобно явному методу Эйлера для вычисления адвекции, этот метод неустойчив для больших величин δt и ν [12]. Поэтому, следуя работе Дж. Стама, снова используем неявную схему для уравнения (11):

$$\left(\mathbf{I} - \nu \delta t \nabla^2\right) \mathbf{u}(\mathbf{x}, t + \delta t) = \mathbf{u}(\mathbf{x}, t), \tag{12}$$

где $\mathbf{I}$ — единичная матрица. Эта схема устойчива для произвольного шага по времени и любой вязкости. Это уравнение есть уравнение Пуассона для скорости. Напомним, что использование декомпозиции Гельмгольца-Ходжа заканчивается уравнением Пуассона для давления. Оба этих уравнения могут быть решены итерационным методом релаксации.

Решение уравнения Пуассона. Необходимо решить два уравнения Пуассона: для давления и для диффузии вязкости. Уравнения Пуассона хорошо изучены. Используем итерационный метод решения, начиная с начального приближения, улучшаем решение на каждой итерации.

Дискретный аналог уравнения Пуассона является системой линейных алгебраических уравнений $\mathbf{Ax} = \mathbf{b}$, где $\mathbf{x}$ – вектор величин, для которых ищем решение (p или $\mathbf{u}$ в нашем случае), $\mathbf{b}$ – вектор констант, $\mathbf{A}$ – матрица. В нашем случае $\mathbf{A}$ представляет оператор Лапласа ∇^2, так что нет необходимости хранить $\mathbf{A}$ как матрицу. Итерационный метод решения стартует с начального приближения решения $\mathbf{x}^{(0)}$, и на каждом шаге k вычисляется новое улучшенное решение $\mathbf{x}^{(k)}$. Надстрочным индексом указывается номер итерации. Пожалуй, самым простым итерационным методом является итерационный метод Якоби. Вид итерации Якоби для СЛАУ можно найти в [3].

Более изощренные методы, такие как метод сопряженных градиентов и многосеточный метод, сходятся быстрее, но здесь используется метод Якоби из-за простоты реализации. Другие методы можно найти в литературе [1, 4, 9].

Уравнения (7) и (12) выглядят по-разному, но оба могут быть дискретизированы, используя выражение (3), и записаны в виде

$$x_{i,j}^{(k+1)} = \frac{x_{i-1,j}^{(k)} + x_{i+1,j}^{(k)} + x_{i,j-1}^{(k)} + x_{i,j+1}^{(k)} + \alpha b_{i,j}}{\beta}, \tag{13}$$

где α и β – константы. Значения x, b, α и β различны для двух уравнений. В уравнении Пуассона для давления x представляет p, b представляет $\nabla \cdot \mathbf{w}$, $\alpha =$

$-(\delta x)^2$ и $\beta = 4$. Для уравнения диффузии вязкости как x, так и b представляют $\mathbf{u}$, $\alpha = (\delta x)^2/(\nu\delta t)$ и $\beta = 4 + \alpha$.

Приведение двух уравнений к одному виду позволяет использовать один исходный код для решения обоих уравнений. Чтобы решить уравнения, просто запускаем множество итераций, в которых применяется формула (13) к каждой ячейке сетки. При этом используются результаты предыдущей итерации $\mathbf{x}^{(k)}$ для вычисления следующей $\mathbf{x}^{(k+1)}$. Так как итерационный метод Якоби сходится медленно, то необходимо выполнить много итераций. Поскольку итерации метода Якоби просты в исполнении на GPU, то можно выполнить много итераций в течение короткого момента времени.

Начальные и граничные условия. Любая задача с дифференциальными уравнениями, определенная в конечной области, требует задания граничных условий. Граничные условия определяются исходя из предположения о способе изменения значений на границе области моделирования. Кроме того, для решения задачи, меняющейся со временем, необходимо задать начальные условия. В нашем случае в начальный момент времени флюид имеет нулевую скорость и нулевое давление в каждой точке. Граничные условия требуют дальнейшего обсуждения.

На каждом шаге решаются уравнения для двух величин — скорости и давления, поэтому необходимы граничные условия для обоих уравнений. Поскольку флюид моделируется на прямоугольной сетке, то предполагается, что флюид находится внутри ящика и не может протечь сквозь стороны ящика. Для скорости на границе используем условие непротекания флюида сквозь границы, что задается стремлением скорости к нулю на границах. Корректное решение уравнения Пуассона для давления требует граничного условия Неймана: $\partial p/\partial n = 0$. Это означает, что на границе, скорость изменения давления в направлении нормали к границе равна нулю. Снова вернемся к граничным условиям в конце пункта 3.

3. Реализация

Теперь, когда понятен способ решения задачи, можно перейти к реализации. Приведем псевдокод алгоритма вычисления одного шага по времени. Переменные u и p предназначены для хранения поля скорости и поля давления соответственно.

```
// применим последовательно 3 оператора из уравнения (9).
 u = advect(u);
 u = diffuse(u);
 u = addForces(u);
// затем применим к результату оператор проекции.
 p = computePressure(u);
 u = subtractPressureGradient(u, p);
```

На практике необходимо использовать дополнительную переменную для выполнения операций. Например, псевдокод шага вычисления адвекции:

```
uTemp = advect(u);
swap(u, uTemp);
```

Этот псевдокод не содержит специфических деталей реализации. Фактически этот псевдокод применим для реализации и на CPU, и на GPU. Наша основная цель выполнить все шаги на GPU. Вычисление такого рода на GPU имеют свою специфику, поэтому проведем некоторые аналогии между действиями при моделировании флюидов на CPU и на GPU.

3.1. CPU–GPU-аналогии

Основой любого компьютера являются модели памяти и процессора, следовательно, любое приложение должно учитывать способы представления данных и вычислений. Рассмотрим различия между CPU и GPU в этом отношении.

Текстуры = Массивы. Рассматриваемая модель флюида представляет данные на двумерной сетке. Естественное представление этой сетки на CPU — массив. Аналог массива на GPU — текстура. Хотя текстуры не такие гибкие, как массивы, но их гибкость постоянно улучшается с развитием графических аппаратных средств. Текстуры на современных GPU поддерживают все основные операции, необходимые для осуществления моделирования флюида. Поскольку текстуры обычно имеют три или четыре цветовых канала, то они обеспечивают естественное представление векторных типов данных от двух до четырех компонент. Кроме того, многочисленные скалярные поля могут быть сохранены в одной текстуре. Основная операция — чтение из массива (или памяти), которая выполняется с использованием операции просмотра текстуры. Таким образом, аналогом индекса массива является координата в текстуре на GPU. Необходимы, по крайней мере, две текстуры для представления состояния флюида: одна для скорости и одна для давления. Для отображения потока необходима дополнительная текстура, которая содержит величины, перемещаемые флюидом.

Тело цикла = Фрагмент программы. При реализации на CPU в цикле выполняется множество шагов алгоритма, используя пару вложенных циклов для повторения операций над каждой ячейкой в сетке. В каждой ячейке сетки выполняются одинаковые вычисления. GPU не имеет возможности выполнять внутренний цикл над каждым текселем (элемент текстуры) в текстуре. Тем не менее, фрагментный конвейер предназначен для выполнения одинаковых вычислений на каждом фрагменте. Программист представляет это, как будто есть отдельный процессор для каждого фрагмента, и все фрагменты обновляются одновременно. На языке параллельного программирования эта модель известна как SIMD (один поток команд - много потоков данных). Таким образом, аналог обработки вложенных циклов над массивом — фрагментная программа на GPU, применяемая к каждому фрагменту методом SIMD.

Обратная связь = Коррекция текстуры. В пункте 2.4 описано, как используется итерация Якоби для решения уравнения Пуассона. Этот итерационный метод использует результат итерации как входные данные для следующей итерации. Такая обратная связь обычна для численных методов. При реализации на CPU обратная связь тривиально осуществляется с использованием переменных и массивов, к которым имеется доступ на чтение и запись. Фрагментный процессор на GPU всегда записывает результат в буфер кадров. Кадровый буфер можно представить себе как двумерный массив, который не может быть напрямую прочитан. Есть два пути получить содержание кадрового буфера в текстуру, которая может быть прочитана:

- *скопировать в текстуру* (CTT) — содержание кадрового буфера копируется в текстуру;
- *рендер в текстуру* (RTT) — используется текстура как кадровый буфер, куда GPU может напрямую записывать.

Функции CTT и RTT одинаково хороши. Выбор функции зависит от конкретной реализации. Далее считается, что реализована операция записи в текстуру.

Ранее упоминалось, что на практике каждые из пяти шагов алгоритма заносят результат во временную сетку, затем выполняется перестановка сеток (свопинг). Функция RTT требует использования двух текстур для осуществления обратной связи, поскольку результаты чтения из текстуры не определены, пока происходит рендеринг. Перестановка в этом случае является просто обменом идентификаторов текстур. Следовательно, временные затраты функции RTT постоянны. С другой стороны, функция CTT требует только одной текстуры. Кадровый буфер используется как временная сетка, и перестановка может быть выполнена копированием данных из кадрового буфера в текстуру. Временные затраты этого копирования пропорциональны размеру текстуры.

3.2. Элементарные операции

Шаги моделирования поделим на элементарные операции. Каждая такая операция включает в себя обработку одного или более фрагментов (зачастую всех) в кадровом буфере активной фрагментной программой, завершаемой обновлением текстуры. Обработка фрагмента управляется рендерингом геометрических примитивов. Используемые в этом приложении примитивы просты: либо прямоугольник, либо отрезок.

Существует два типа фрагментов для обработки в каждой элементарной операции: внутренние фрагменты и граничные фрагменты. Наша двумерная сетка имеет границу, составленную из периметра, шириной в одну ячейку. Обычно вычисления различны для ячеек внутри и на границе. Для обновления внутренних фрагментов рисуется прямоугольник, покрывающий весь кадровый буфер, за исключением границы, шириной в одну ячейку. Граница выводится с помощью четырех отрезков. Реализуются отдельные фрагментные программы для внутренних и граничных ячеек. Смотри рисунок 4.

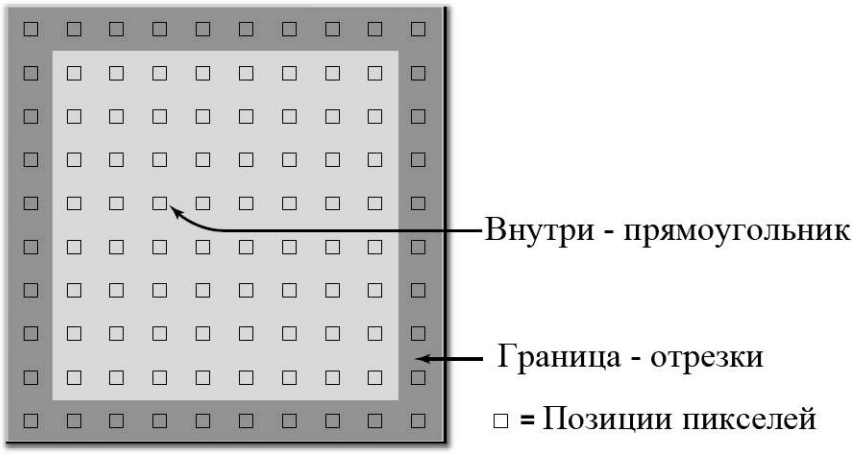

Рис. 4. Примитивы, используемые для обновления внутренних и граничных ячеек сетки

3.3. Реализация фрагментных программ

Теперь мы знаем шаги алгоритма, представление данных и как выполняются элементарные операции. Приступим к написанию фрагментных программ для выполнения вычислений в каждой ячейке.

Перемещение потока. Реализация фрагментной программы для адвекции приведена в листинге 1, которая почти повторяет уравнение (10). Имеется одно незначительное отличие. Поскольку координаты текстуры изменяются в диапазоне отличном от координат области моделирования (координаты текстуры — в диапазоне $[0, N]$, где N — разрешение сетки), необходимо масштабировать скорость в пространство сетки. Это отражено в коде Cg с помощью умножения локальной скорости на параметр rdx, который представляет обратную величину шага сетки δx. Способ заворачивания текстуры должен быть установлен на CLAMP_TO_EDGE, чтобы значения, выходящие за пределы диапазона $[0, N]$, были прикреплены к граничным текселям. Граничные условия корректно обрабатывают эти тексели, как будет описано ниже.

Листинг 1. *Фрагментная программа для адвекции*

```
void advect(float2 coords   : WPOS,   // координаты сетки
   out float4 xNew : COLOR,  // результат адвекции qty
   uniform float timestep,
   uniform float rdx,        // 1 / dx
   uniform samplerRECT u,    // входное поле скорости
   uniform samplerRECT x)    // qty для адвекции
{
  // учитывая поле скорости, делаем "шаг назад"
   float2 pos = coords - timestep * rdx * f2texRECT(u, coords);

  // интерполируем и записываем результат
  xNew = f4texRECTbilerp(x, pos);
}
```

В этом коде параметр u является текстурой поля скорости, а x — поле, которое будет получено после адвекции. Это может быть скоростью или любой другой величиной, например концентрация краски. Функция f4texRECTbilerp() выполняет билинейную интерполяцию четырех текселей, ближайших к точке с вычисленными текстурными координатами. Поскольку современные GPU не поддерживают автоматическую билинейную интерполяцию в текстурах со значениями с плавающей точкой, то необходимо осуществить это с помощью данной функции.

Диффузия вязкости. Описание итерации метода Якоби, приведенное выше в пункте 2.4, реализовано фрагментной программой в листинге 2.

Листинг 2. *Фрагментная программа итерации Якоби для решения уравнения Пуассона*

```
void jacobi(half2 coords   : WPOS,   // координаты на сетке
   out
   half4 xNew : COLOR,  // результат
   uniform half alpha,
   uniform half rBeta,      // 1/ beta
   uniform samplerRECT x,   // вектор x (Ax = b)
   uniform samplerRECT b)   // вектор b (Ax = b)
{
  // левая, правая, нижняя и верхняя ячейки к x
  half4 xL = h4texRECT(x, coords - half2(1, 0));
  half4 xR = h4texRECT(x, coords + half2(1, 0));
  half4 xB = h4texRECT(x, coords - half2(0, 1));
  half4 xT = h4texRECT(x, coords + half2(0, 1));

  // ячейка b из центра
   half4 bC = h4texRECT(b, coords);

  // выполнение итерации Якоби
  xNew = (xL + xR + xB + xT + alpha * bC) * rBeta;
}
```

Заметим, что параметр rBeta — это обратная величина к β из формулы (13). Для решения уравнения диффузии устанавливается alpha$= (\delta x)^2/(\nu\delta t)$, rBeta$= 1/\big(4 + (\delta x)^2/(\nu\delta t)\big)$, а также параметры x и b на текстуру поля скорости. Затем выполняется некоторое количество итераций (обычно от 20 до 50, большее количество можно использовать, чтобы уменьшить ошибку).

Приложение сил. Самый простой шаг алгоритма – это вычисление ускорения под действием внешних сил. В моделирующем приложении импульс внешней силы можно прикладывать к флюиду посредством нажатия и перемещения мыши. Для реализации этого рисуем пятно в текстуре поля скоростей в позиции

щелчка мыши. Цвет пятна кодирует силу и направление импульса: красный канал содержит величину вдоль оси x, а зеленый канал содержит величину вдоль оси y. Пятно представляется двумерной Гауссовской «шапочкой».

Используем фрагментную программу для вычисления расстояния до точки приложения импульса. Затем добавляем величину $\mathbf{c}$ к цвету

$$\mathbf{c} = \mathbf{F}\delta t \exp\left(\frac{(x - x_p)^2 + (y - y_p)^2}{r}\right),$$

здесь $\mathbf{F}$ — сила, вычисленная в зависимости от расстояния и направления перемещения мыши, r — заданный радиус импульса, (x, y) и (x_p, y_p) — позиции фрагмента и импульса (щелчка мыши) в координатах окна соответственно.

Проекция. В начале этого пункта указывалось, что оператор проекции реализуется за две операции: решение уравнения Пуассона для поля давления p и вычитание градиента p из промежуточного поля скорости. Для этого потребуются три фрагментные программы: вышеупомянутая программа итерации Якоби, программа для вычисления дивергенции промежуточного поля скорости, программа для вычитания градиента p из промежуточного поля скорости.

Программа для вычисления дивергенции приведена в листинге 3. В качестве входных параметров для нее требуются промежуточное поле скорости w и половина обратной величины шага сетки halfrdx. Она вычисляет дивергенцию согласно конечно-разностной формуле, приведенной в таблице 1.

Результат дивергенции записывается во временную текстуру, которая затем используется как входной параметр b в программе итерации Якоби. Параметр x в программе итерации Якоби устанавливается в текстуру поля давления, которая первоначально имеет все нулевые значения (другими словами, нуль используется как начальное предположение для поля давления). Параметры alpha и rBeta установлены как $-(\delta x)^2$ и $1/4$ соответственно.

Для достижения хорошей сходимости решения будем использовать от 40 до 80 итераций Якоби. Изменение количества итераций Якоби будет влиять на точность моделирования. Если количество итераций меньше 20, то ошибка уже заметна. Использование большего количества итераций приводит к более подробной детализации и повышению точности, но требует больших временных затрат для вычисления. После завершения итерации Якоби текстура поля давления передается как параметр p в следующую программу, которая вычисляет градиент p согласно определению из таблицы 1 и вычитает его из промежуточного поля скорости, хранящегося в текстуре w. Смотри листинг 4.

Листинг 3. *Фрагментная программа вычисления дивергенции*

```
void divergence(half2 coords  : WPOS,   // координаты на сетке
   out
   half4 div : COLOR,  // дивергенция
   uniform half halfrdx,   // 0.5 / шаг сетки
   uniform samplerRECT w)  // векторное поле
{
```

```
  half4 wL = h4texRECT(w, coords - half2(1, 0));
  half4 wR = h4texRECT(w, coords + half2(1, 0));
  half4 wB = h4texRECT(w, coords - half2(0, 1));
  half4 wT = h4texRECT(w, coords + half2(0, 1));

  div = halfrdx * ((wR.x - wL.x) + (wT.y - wB.y));
}
```

Листинг 4. *Фрагментная программа вычитания градиента*

```
void gradient(half2 coords   : WPOS,   // координаты на сетке
   out half4 uNew : COLOR,  // новая скорость
   uniform half halfrdx,    // 0.5 / шаг сетки
   uniform samplerRECT p,   // поле давления
   uniform samplerRECT w)   // поле скорости
{
  half pL = h1texRECT(p, coords - half2(1, 0));
  half pR = h1texRECT(p, coords + half2(1, 0));
  half pB = h1texRECT(p, coords - half2(0, 1));
  half pT = h1texRECT(p, coords + half2(0, 1));

  uNew = h4texRECT(w, coords);
  uNew.xy -= halfrdx * half2(pR - pL, pT - pB);
}
```

Граничные условия. В пункте 2.4 определено, что наш флюид находится в «ящике» и граничные условия задаются равными нулю для скорости и граничные условия Неймана для давления. В пункте 3.2 показано, что граничные условия реализуются по периметру шириной в один пиксель сетки. Граничные условия обновляются при рисовании отрезков, используя фрагментную программу для соответствующих данных.

Сначала найдем, как данная дискретизация сетки влияет на вычисление граничных условий. Из условия отсутствия протекания на границе следует, что скорость равняется нулю на границе, а условие Неймана для давления требует равенства нулю производной давления по нормали к границе. Граница проходит между граничной ячейкой и ближайшей внутренней ячейкой, но значения на сетки определены в центрах ячеек. Следовательно, необходимо вычислять граничные значения так, чтобы среднее между значениями в двух смежных ячейках к границе удовлетворяло граничному условию.

Например, для скорости на левой границе имеем

$$\frac{\mathbf{u}_{0,j} + \mathbf{u}_{1,j}}{2} = 0, \quad j \in [0, N], \tag{14}$$

где N — разрешение сетки. Для удовлетворения этого условия необходимо установить $\mathbf{u}_{0,j}$, равное $-\mathbf{u}_{1,j}$. Уравнение давления рассматривается аналогично.

Используя разностную аппроксимацию производной, получим

$$\frac{p_{1,j} - p_{0,j}}{\delta x} = 0. \tag{15}$$

Из решения этого уравнения для $p_{0,j}$ видно, что необходимо установить значение величины давления на границе, равной величине давления во внутренней смежной ячейке к границе.

Можно использовать простую фрагментную программу как для давления, так и для скорости на границе, смотри листинг 5.

Листинг 5. *Фрагментная программа для граничных условий*

```
void boundary(half2 coords : WPOS,    // координаты на сетке
   half2 offset : TEX1,    // смещение вдоль границы
   out half4 bv : COLOR,   // выходное значение
   uniform half scale,     // параметр масштабирования
   uniform samplerRECT x)  // поле значений
{
  bv = scale * h4texRECT(x, coords + offset);
}
```

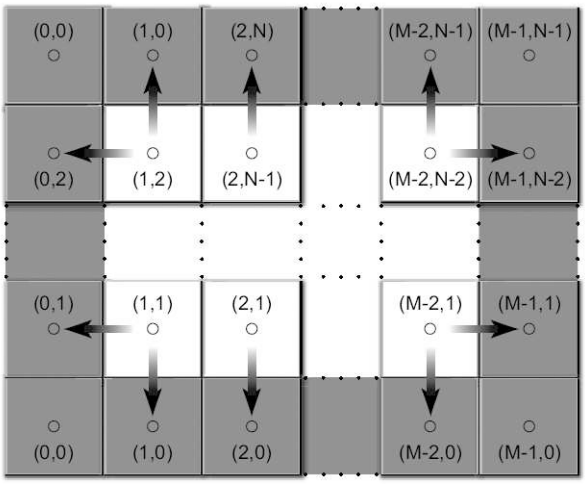

Рис. 5. Граничные условия на MxN-сетке

На рисунке 5 демонстрируется работа этой программы. Параметр x представлен текстурой (поле скорости или давления), из которой прочитывают внутренние величины. Параметр offset задает корректное смещение к внутренней ячейке, смежной с текущей граничной ячейкой. Параметр coords содержит позицию в координатах текстуры обрабатываемого фрагмента, так что, прибавляя смещение к нему, получаем соседний тексель. На каждой границе устанавливается свое смещение для регулирования координат текстуры для текселей, находящихся внутри от границы. Для левой границы установлено значение смещения $(1, 0)$, чтобы найти соседний тексель справа от границы; для нижней

границы значение смещения — $(0, 1)$; и так далее. Параметр scale может быть использован для масштабирования значения, которое переносится на границу. Для граничных условий для скорости масштабный коэффициент равен -1, а для давления 1, для того чтобы правильно реализовать уравнения (14) и (15) соответственно.

3.4. Три измерения

Представленная реализация моделирования флюидов — двумерна. Её легко расширить до трехмерной. Вид уравнений существенно не изменится, но их количество увеличится, поскольку вектор скорости $\mathbf{u} = (u, v, w)$ будет трехмерным. В связи с этим фрагментные программы должны быть переписаны. Например, в двумерном пространстве ячейка имеет четырех соседей, а в трехмерном — шесть.

Большое изменение касается представления векторных и скалярных полей. Можно воспользоваться трехмерными текстурами, но аппаратные средства не поддерживают трехмерные текстуры со значениями с плавающей точкой. Поэтому можно уложить трехмерную текстуру в двумерную. Например, сетка 32x32x32 ячейки укладывается мозаикой в двумерную текстуру 256x128 ячеек. Этот метод называют плоскими 3D текстурами, более подробно он описан в [5].

4. Примеры

Приведем различные результаты моделирования флюидов на CPU и GPU.

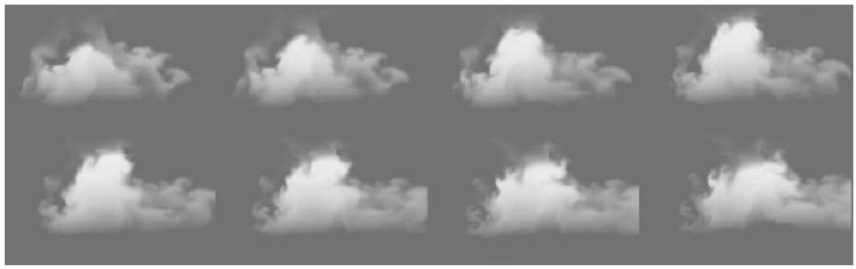

Рис. 6. Последовательность кадров моделирования облаков

Первый пример (рис. 6) представляет результат моделирования двумерного изображения облаков, реализованное М. Харрисом [7] на основе уравнения Навье-Стокса. Также в моделировании учитываются сила выталкивания и сила торможения турбулентности. Решение уравнений основано на декомпозиции Гельмгольца-Ходжа. На рис. 6 приведена последовательность кадров моделирования на сетке с разрешением 128x128 ячеек.

На рис. 7 приведен кадр имитации полета с отображением облаков. Облака генерируются в виде множества частиц различных форм и плотностей. Наложение изображений облаков обеспечивает вполне реальное воспроизведение объема. Подробнее можно ознакомиться в работе М. Харриса и А. Ластра [8].

Рис. 7. Имитация трехмерных облаков

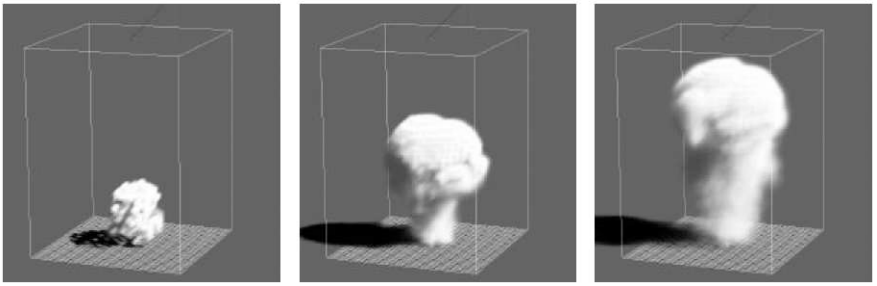

Рис. 8. Имитация распространения дыма

В работе Дж. Стама [12] эффективность метода устойчивых флюидов продемонстрирована на примере распространения дыма (рис. 8). Уравнения Навье–Стокса были реализованы на трехмерной сетке. Вычисления производились на CPU.

Дж. Крюгер и Р. Вестерманн в работе [9] демонстрируют модель двумерной поверхности воды как пример использования предложенного авторами алгоритма решения систем алгебраических уравнений на GPU (рис. 9).

В работе В. Ли, Ж. Фана, К. Вейя и А. Кауфмана [10] представлена модель потока флюида, основанная на сеточном методе Больцмана (Lattice Boltzmann Method, LBM). Рисунок 10 показывает результаты моделирования потока, обтекающего различные препятствия: неподвижная ваза, движущаяся сфера, плывущая медуза. Авторы модели отмечают, что при движении медуза деформирует свое тело, поэтому изменяется граница между жидкостью и телом медузы. Линиями на рисунке представлены направления движения потока. Все этапы

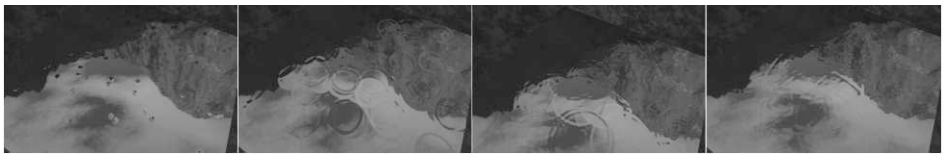

Рис. 9. Последовательность кадров моделирования поверхности воды

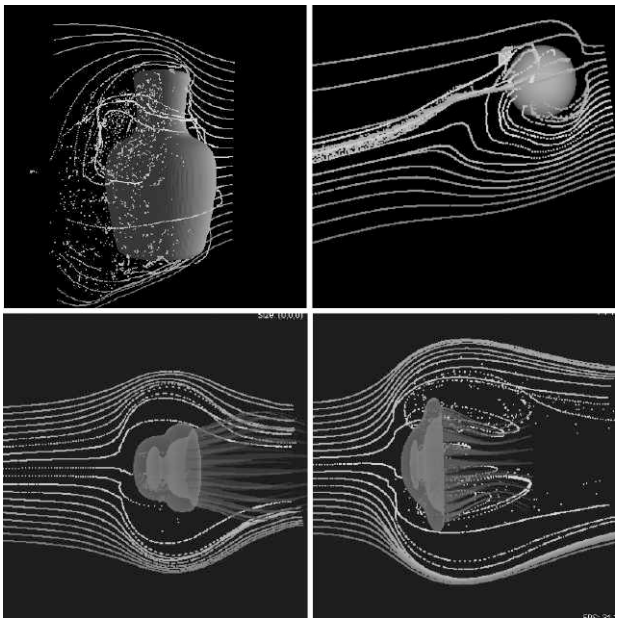

Рис. 10. Имитация поля потока обтекания предметов

вычисления и отображения выполнялись на GPU в реальном времени.

Литература

1. Bolz, J. Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid / J. Bolz, I. Farmer, E. Grinspun, P. Schröder // Proceedings of SIGGRAPH. – 2003. – P. 917–924.
2. Chorin, A.J. A Mathematical Introduction to Fluid Mechanics / A.J. Chorin, J.E. Marsden. – Springer-Verlag, 1993.
3. Golub, G.H. Matrix Computations / G.H. Golub, C.F. van Loan. – The Johns Hopkins University Press, 1996.
4. Goodnight, N. A Multigrid Solver for Boundary Value Problems Using Graphics Hardware / N. Goodnight, C. Woolley, G. Lewin, D. Luebke, G. Humphreys // Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware. – 2003. – P. 102–111.
5. Harris, M.J. Simulation of Cloud Dynamics on Graphics Hardware / M.J. Harris, W.V. Baxter, T. Scheuermann, A. Lastra // Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware. – 2003. P. 92–101.
6. Harris, M.J. Fast Fluid Dynamics Simulation on the GPU / M.J. Harris // GPU Gems, edt. Randima Fernando. – Addison-Wesley, 2004. – P. 637–665.
7. Harris, M.J. Real-Time Cloud Simulation and Rendering / M.J. Harris // University of North Carolina Technical Report TR03-040. – 2003.
8. Harris, M.J. Real–time cloud rendering / M.J. Harris, A. Lastra // Proceedings of Eurographics. – 2001. – P. 76–84.
9. Krüger, J. Linear Algebra Operators for GPU Implementation of Numerical Algorithms / J. Krüger, R. Westermann // Proceedings of SIGGRAPH. – 2003.

10. Li W. Flow Simulation with Complex Boundaries / W. Li, Z. Fan, X. Wei, A. Kaufman // GPU Gems II: Programming Techniques for High-Performance Graphics and General-Purpose Computation, edt. Matt Pharr. – Addison-Wesley, 2005. – P. 747–764.
11. Stam, J. Stable Fluids / J. Stam // Proceedings of SIGGRAPH. – 1999. – P. 121–128.
12. Stam, J. Real-time fluid dynamics for games / J. Stam // Proceedings of the Game Developers Conference. – 2003.

Математические структуры и моделирование
2009, вып. 20, с. 74–77

УДК 530.12:531.51

СЕМЕЙСТВО Ω-ОБРАЗНЫХ КРИВЫХ, МОДЕЛИРУЮЩЕЕ ОТДЕЛЕНИЕ СФЕРЫ S^1

Е.В. Палешева

В работе приводится построение семейства Ω-образных C^1-кривых, с помощью которых можно построить модель отделения от кривой второго порядка сферы S^1. Каждая кривая образована склеиванием окружности с гиперболой. В точках склеивания вторая производная терпит разрыв.

Введение

Ранее в работах [1–4] была предложена процедура создания 4-мерной кротовой норы. Для этого необходимо произвести разрыв замкнутого 3-мерного пространства на два куска, 4-мерное пространство-время становится 2-связным. В [5] было проведено исследование вопроса, связанного с образованием в пространстве-времени 4-мерной ручки (кротовой норы) с топологической точки зрения. Так как до сих пор не были получены решения уравнений Эйнштейна, содержащие 4-мерные кротовые норы, мы попытаемся восполнить этот пробел. В данной работе приводится семейство кривых, с помощью которого можно построить модель разрыва одномерного пространства на два несвязных куска. На завершающем этапе будет получена прямая, склеенная с окружностью в одной точке. При этом первоначальной кривой является прямая линия. Указанное семейство мы назовем Ω-образным. На этапе построения Ω-образного семейства приводится семейство ω-кривых, в дальнейшем будем их использовать для моделирования аналогичного разрыва 2-мерных и 3-мерных пространств, на которых определим метрический тензор. Для построения семейства Ω-образных кривых произведем склейку гиперболы и окружности в заданной области.

1. Семейства гипербол и касательных окружностей

Рассмотрим кривую второго порядка $G(\lambda, \mu; x, y) = 0$, $y > 0$, где

$$G(\lambda, \mu; x, y) = (\lambda^2 - \mu^2)y^2 + 2\lambda\mu xy - \lambda^4. \tag{1}$$

Нетрудно проверить, что данная кривая является гиперболой. Ее асимптоты представлены уравнениями $y = 0$, $y = 2\mu x/\lambda(\mu^2 - 1)$, фокальная ось задается

Copyright © 2009 **Е.В. Палешева.**
Омский государственный университет.
E-mail: palesheva@cmm.univer.omsk.su

соотношением $y = \lambda x/\mu$, а точка ее пересечения с кривой имеет координаты $(\lambda\mu/\sqrt{\lambda^2+\mu^2},\ \lambda^2/\sqrt{\lambda^2+\mu^2})$. Следует заметить, что данная гипербола может быть также представлена как $x_{\lambda\mu}(y)$ однозначным образом для любого $y > 0$:

$$x_{\lambda\mu}(y) = \frac{\lambda^4 - (\lambda^2-\mu^2)y^2}{2\mu\lambda y}, \quad y > 0. \tag{2}$$

При этом в точке $(\lambda\sqrt{\mu^2-\lambda^2}/\mu,\ \lambda^2/\sqrt{\mu^2-\lambda^2})$ кривая $x_{\lambda\mu}(y)$ имеет минимум при условии $\lambda < \mu$ и значение указанной функции в данной точке наименьшее. Других экстремумов функция (2) не имеет.

Рассмотрим окружность $S^1(\lambda,\mu,\nu;x,y) = 0$, где

$$S^1(\lambda,\mu,\nu;x,y) = x^2 + \left(y - \frac{(\mu^2+\lambda^2)^2\nu^4 - \lambda^8}{4\mu^2\lambda^2\nu^3}\right)^2 - \tag{3}$$
$$- \frac{(\lambda^4 + (\mu^2-\lambda^2)\nu^2)^2}{4\mu^2\lambda^2\nu^2} - \frac{\left((\mu^2-\lambda^2)^2\nu^4 - \lambda^8\right)^2}{16\mu^4\lambda^4\nu^6}.$$

Элементарными вычислениями проверяется, что данная окружность является касательной к рассмотренной гиперболе в точке $M_\nu = (x_{\lambda\mu}(\nu),\nu)$. Следует также заметить, что в точке пересечения гиперболы со своей фокальной осью касательная окружность принимает вид: $x^2 + y^2 = \lambda^2$.

2. Семейство Ω-образных кривых

Вначале построим семейство кривых, которые будем называть ω-кривыми, оно будет определено при $x \geq 0$, $y > 0$. В дальнейшем семейство двумерных поверхностей, моделирующее отрыв сферы S^2 от плоскости, можно получить вращением соответствующих ω-кривых вокруг оси OY.

2.1. Построение ω-кривых

Определим функцию $\omega(\lambda,\mu,\nu;x,y)$ по следующему правилу:

$$\omega(\lambda,\mu,\nu;x,y) = \begin{cases} G(\lambda,\mu;x,y), & \text{если } 0 < y \leq \nu, \\ S^1(\lambda,\mu,\nu;x,y), & \text{если } y > \nu. \end{cases} \tag{4}$$

Здесь $G(\lambda,\mu;x,y)$ и $S^1(\lambda,\mu,\nu;x,y)$ функции (1) и (3) соответственно. При этом $x \geq 0$. Тогда неявная функция $\omega(\lambda,\mu,\nu;x,y) = 0$ определит ω-кривую при фиксированных параметрах λ, μ и ν. При условии $\lambda,\mu,\nu > 0$ и $x,y > 0$ данные кривые непрерывны по построению вместе со своими первыми производными. Вторая производная в точке склеивания является разрывной. Вид приведенной ω-кривой при $\mu = 0$ получаем предельным переходом. Если $\mu \to 0$, то функция $\omega(\lambda,\mu,\nu;x,y) = 0$ вырождается в прямую $y = \lambda$.

Построение семейства кривых, с помощью которых в дальнейшем можно получить семейство поверхностей, моделирующих отрыв сферы S^2 от плоскости, проводится путем изменения параметров μ, λ, ν в три этапа.

(1) **Выдавливание полуокружности из прямой.** Положим $\lambda = \lambda_0 > 0$ и $\nu = \lambda_0^2/\sqrt{\lambda_0^2+\mu^2}$, т.е. окружность[1] и гипербола касаются друг друга в точке, лежащей на фокальной оси гиперболы. Уравнение кривой принимает вид

$$\omega\left(\lambda_0, \mu, \frac{\lambda_0^2}{\sqrt{\lambda_0^2+\mu^2}}; x, y\right) = 0, \quad \mu \in [0, \mu_1]. \tag{5}$$

Параметр μ меняем от $\mu_0 = 0$ (график соответствует прямой) до значения $\mu = \mu_1 > \lambda_0$, при котором $\nu = \lambda_0^2/\sqrt{\lambda_0^2+\mu_1^2}$.

(2) **Образование «перемычки».** Увеличиваем ν от $\nu_1 = \lambda_0^2/\sqrt{\lambda_0^2+\mu_1^2}$ до $\nu_2 = 2\lambda_0^2/\sqrt{\mu_1^2-\lambda_0^2}$ при постоянных $\lambda = \lambda_0$ и $\mu = \mu_1$. На этом этапе центр окружности смещается вверх по оси OY. Радиус окружности сначала уменьшается до значения $\lambda_0\sqrt{\mu_1^2-\lambda_0^2}/\mu_1^2$ (которое соответствует касанию в точке экстремума), а затем увеличивается. При этом точка касания лежит выше точки экстремума, таким образом образуется перемычка. Конечное значение ν_2 в два раза больше ординаты точки экстремума гиперболы[2]. Кривая имеет вид

$$\omega\left(\lambda_0, \mu_1, \nu; x, y\right) = 0, \quad \nu \in [\nu_1, \nu_2]. \tag{6}$$

(3) **Стягивание «перемычки» в точку.** Уменьшаем λ от λ_0 до $\lambda_1 = 0$ при $\mu = \mu_1$, а также $\nu = 2\lambda^2/\sqrt{\mu_1^2-\lambda^2}$. Радиус окружности и абсцисса точки касания уменьшаются. Уравнение кривой при $\lambda_1 \neq 0$ имеет вид

$$\omega\left(\lambda, \mu_1, \frac{2\lambda^2}{\sqrt{\mu_1^2-\lambda^2}}; x, y\right) = 0, \quad \lambda \in [\lambda_0, \lambda_1]. \tag{7}$$

Осуществляя предельный переход кривой (7) при $\lambda \to 0$ получим

$$\begin{cases} y = 0, & \text{если } y = 0, \\ x^2 + \left(y - \frac{15}{32}\,\mu_1\right)^2 = \left(\frac{15}{32}\,\mu_1\right)^2, & \text{если } y > 0. \end{cases}$$

Напомним, что здесь везде $x \geq 0$. Таким образом, получена полуокружность, склеенная с полупрямой $y = 0$ в точке с координатами $(0, 0)$.

2.2. Построение Ω-образных кривых

Симметрично продолжая функцию (4) на область $x < 0$, получим

$$\Omega(\lambda, \mu, \nu; x, y) = \begin{cases} G(\lambda, \mu; x, y), & \text{если } 0 < y \leq \nu \text{ и } x \geq 0, \\ S^1(\lambda, \mu, \nu; x, y), & \text{если } y > \nu, \\ G(\lambda, \mu; -x, y), & \text{если } 0 < y \leq \nu \text{ и } x < 0. \end{cases}$$

[1] На этом этапе окружность описывается уравнением $x^2 + y^2 = \lambda_0^2$.

[2] В качестве ν_2 можно выбрать любую величину больше ординаты точки экстремума.

В качестве семейства Ω-образных кривых рассмотрим соответственно симметричное продолжение построенного в предыдущем параграфе семейства ω-кривых на область $x < 0$. При этом все этапы изменения параметров λ, μ и ν остаются без изменения.

В итоге построенное семейство Ω-образных кривых при непрерывном изменении параметров описывает следующие действия. Из прямой $y = \lambda > 0$ постепенно выдавливается с образованием «перемычки» окружность. В результате получается окружность

$$x^2 + \left(y - \frac{15}{32}\,\mu_1\right)^2 = \left(\frac{15}{32}\,\mu_1\right)^2,$$

склеенная с прямой $y = 0$ в точке с координатами $(0, 0)$. Теперь останется только оторвать окружность от данной прямой.

Заключение

В данной статье нами было построено семейство Ω-образных C^1-гладких кривых, моделирующих отделение сферы S^1 от прямой. «Перемычка» была стянута в точку, но отрыв полученной окружности от прямой мы не производили. Полученное в данной работе семейство C^1-гладких ω-кривых можно применить для построения поверхностей, моделирующих отрыв сферы соответствующей размерности. Необходимо только предварительно выполнить вращение вокруг заданной оси. Впоследствии на данных поверхностях можно также определить метрический тензор.

Литература

1. Гуц, А.К. Изменение топологии физического пространства в замкнутой вселенной / А.К. Гуц // Известия вузов. Физика. – 1982. N. 5. – С. 23-26.
2. Гуц, А.К. Нарушение связности физического пространства / А.К. Гуц // Известия вузов. Физика. – 1983. N. 8. – С. 3-6.
3. Гуц, А.К. Теория машины времени / А.К. Гуц //В сб.: Фундаментальная и прикладная математика. – Омск: изд-во ОмГУ, 1994. С. 57-66.
4. Гуц, А.К. Элементы теории времени / А.К. Гуц. – Омск: изд-во Наследие. Диалог-Сибирь, 2004. 364 с.
5. Гуц, А.К. Машина времени, разрывы пространства и 4-мерные кротовые норы / А.К. Гуц, Е.В. Палешева // Вестник Красноярского государственного университета. Физико-математические науки. – 2005. – N. 7. – С. 138-142.

Математические структуры и моделирование
2009, вып. 20, с. 78–92

УДК 004.588

ЭЛЕМЕНТЫ ПОВТОРНОГО ИСПОЛЬЗОВАНИЯ ДЛЯ ПРОГРАММНЫХ СИСТЕМ УЧЕБНОГО НАЗНАЧЕНИЯ. КОНЦЕПТУАЛЬНОЕ ПРОЕКТИРОВАНИЕ И АНАЛИЗ

С.В. Гусс

В работе рассматриваются элементы повторного использования, выявленные на этапе анализа предметной области, которые могут служить отправной точкой для стадии проектирования в процессе разработки программных систем учебного назначения. Представлена модель элементов предметной области в виде диаграмм в нотации UML. Элементы разделены на семь подгрупп: управления пользователями, обучения, обслуживания, взаимодействия с пользователем, безопасности, коммуникации, учебного материала. Методы исследования, с помощью которых проводились концептуализация и анализ в данной работе, принадлежат объектно-ориентированной парадигме: моделированию и анализу.

Введение

Сегодня, в самом начале XXI века, отчётливо видна тенденция внедрения информационных технологий в систему образования, начало которому было положено примерно в 70-е — 80-е года XX века. Активно идет процесс проникновения информационных технологий в сферу образования, и одним из направлений повышения эффективности их применения является интеграция информационно-коммуникационных технологий и технологий обучения [1], в частности основанных на играх. Программные системы, разрабатывающиеся для сферы образования, призваны увеличить эффективность работы преподавателей и качество усвоения информации обучающимися на всех уровнях образования, от общего до профессионального. Такие системы помимо технической поддержки процесса обучения осуществляют педагогические функции.

Программная система учебного назначения должна не только помогать преподавателям в контроле знаний учащихся, но и в освоении нового учебного материала, тренировать и стимулировать интерес к изучаемому предмету.

Copyright © 2009 **С.В. Гусс.**
Омский государственный университет.
E-mail: InfoGuss@gmail.com

Ключевая идея создания элементов повторного использования в целях исследования автора – повышение эффективности разработки класса программных систем учебного назначения, основанных на играх-головоломках лингвистической направленности. Под эффективностью создания в работе понимается, прежде всего, уменьшение времени разработки приложений, относящихся к классу рассматриваемых.

В случае, когда приложения проектируются таким образом, как отмечено в [2], что «различные их части могут быть использованы многократно», уменьшается стоимость их разработки. Для возможности многократного использования приложение должно состоять из идентифицируемых (распознаваемых) и заменяемых частей. На основании [3] повторное использование – важный аспект, который «оказывает большое влияние на эффективность всех направлений работ и на качество изделий». Основные преимущества повторного использования [4]: повышение надёжности (благодаря тому что элементы повторного использования протестированы и проверены в разных условиях и режимах работы), уменьшение расходов, эффективное использование специалистов, соблюдение стандартов, ускорение разработки (за счёт сокращения времени на разработку и тестирование компонентов).

Полезную информацию о повторном использовании можно подчеркнуть из [5], основные идеи которой состоят в следующем. Библиотека, как коллекция ресурсов служит основным средством повторного использования в сфере разработки программного обеспечения; повторное использование может быть как запланированным, так и незапланированным процессом; в случае запланированного процесса можно говорить о разработке каркасов приложений, т.е. скелетов программ, поддающихся детализации в целях создания нужного приложения; в случаях, когда повторное использование не запланировано, речь идёт о сохранении и повторном использовании кода, что является менее эффективным. Однако библиотеки и каркасы (каркас также включает парадигму потока управления) не единственные элементы повторного использования. Не менее полезными элементами являются модели и образцы (или паттерны, т.е. готовое решение возникшей задачи; большинство паттернов показывают, как оптимально работать с коллекцией классов).

1. Программные системы учебного назначения

Под программной системой учебного назначения следует понимать собранный в одну систему набор программных средств поддержки учебных дисциплин, имеющий предметное содержание и ориентированный на взаимодействие с обучаемым. Основные требования, предъявляемые к таким системам: они должны решать педагогические задачи, должны включать определённый материал по определённым предметным областям. Главные элементы такой системы: графическая оболочка, содержание (текстовые, аудиовизуальные и другие ресурсы) и система управления этим содержанием. Базовая категория пользователей этих компьютерных сред обучения – обучаемые и преподаватели (включая инструкторов и методистов), использующие их в профессиональной деятельности. Одна

из важнейших характеристик таких систем - ориентация на самостоятельную работу обучаемых.

Следует разобраться в потребностях предполагаемых клиентов (потребителей) рассматриваемых систем, коими являются различные образовательные учреждения и пользователи, которые учатся самостоятельно в домашних условиях с помощью персонального компьютера. Попытки ответить на основные вопросы, возникающие на стадии концептуального проектирования, дают следующее:

1. Кто нуждается в таких приложениях, т.е. кто будет конечным потребителем? Ответ такой: это прежде всего образовательные учреждения и домашние пользователи.

2. Какую задачу они решают? Самое главное – они должны решать педагогические задачи, либо помочь в их решении. Т.е. используются в учебном процессе и для самообразования.

3. Где планируется использовать такие приложения? Имеется в виду тип оборудования и коммуникации. Не должно быть сомнений в том, что приоритетное место за многоплатформенностью. В идеале приложения должны функционировать на различных типах компьютерного оборудования: на рабочих станциях и на мобильных системах (ноутбуках, планшетах, карманных компьютерах).

4. Когда они будут нужны? Можно сказать, что уже есть в них потребность. Подобные системы если и не используются в образовательных учреждениях, то к ним, по крайней мере, присматриваются.

5. Почему они нужны? Вопрос касается технико-экономического обоснования, и его правильнее решать для конкретной системы из рассматриваемого класса.

6. Как они будут работать? Вопрос опять же относится к конкретной системе.

На рис. 1, 2, 3 представлены диаграммы прецедентов (использования) обучающих систем.

На рис. 1 изображены сценарии использования обучающей системы непосредственно обучаемыми.

Представленные варианты соответствуют основным педагогическим задачам, решаемым с помощью данных систем. Это, как отмечается в [6]:

1. Знакомство, т.е. первичное ознакомление с изучаемой предметной областью, освоение основных концепций и определений.

2. Подготовка по различным разделам дисциплины с определённым уровнем глубины и детализации.

3. Развитие способностей к специфичным видам деятельности.

Рис. 1. Сценарии использования обучающей системы обучаемым

4. Восстановление знаний и умений, приобретённых в прошлом.

5. Контроль успеваемости, проведение тестов. Можно выделить следующие виды контроля (названия видов взяты из [7]):

   (a) Входной. Необходим для определения уровня знаний перед изучением дисциплины.

   (b) Текущий. Используется при изучении отдельных разделов и их частей конкретной дисциплины.

   (c) Промежуточный. Определяет качество усвоения отдельного раздела.

   (d) Итоговый. Служит для проверки качества усвоения материала по целому курсу (дисциплине).

   (e) Проверка остаточных знаний.

На рис. 2 представлены сценарии общения администратора с системой. Две основных обязанности администратора при работе с обучающими системами:

1. Настройка, конфигурирование системы.

2. Проверка функциональности.

На рис. 3 представлена диаграмма прецедентов для случая общения инструктора с системой. Три типичных случая взаимодействия:

1. Составление задания. Это его создание и регистрация в системе.

2. Настройка системы. Включает назначение заданий обучаемым и открытие доступа обучаемых к учебному материалу и заданиям.

Рис. 2. Сценарии использования обучающей системы администратором

3. Анализ работы обучающегося. Просмотр протокола работы обучающегося (результаты выполнения заданий, затраты времени на изучение и проверку знаний, действия, совершённые за период общения с системой), просмотр модели обучающегося, т.е. результатов интеллектуальной работы системы, а именно выводов, сделанных из протокола работы обучаемых [6].

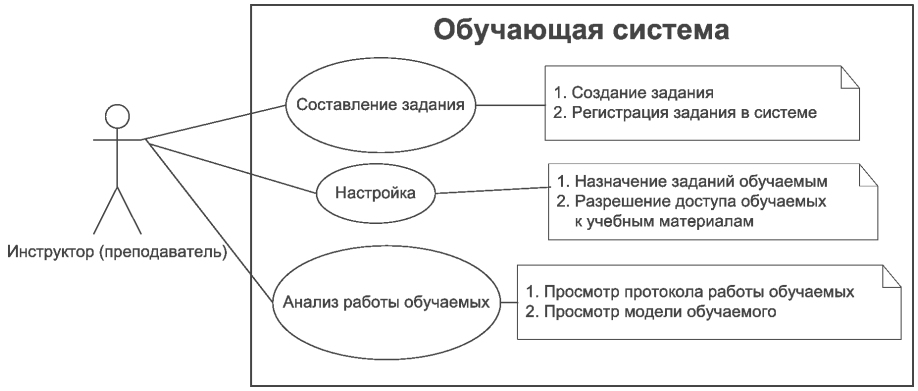

Рис. 3. Сценарии использования системы инструктором (преподавателем)

Представленные сценарии подходят для использования в учебных заведениях для решения основных педагогических задач. Если инструктор не является живым человеком, а представляет собой систему, внешнюю по отношению к рассматриваемой, или является её частью, ответственной за работу с учебным материалом и заданиями, то обучающая система может представлять возможности по самообразованию. В работе [8] отмечается, что «исключительно важное значение придаётся разработке и внедрению эффективных технологий самоконтроля», что говорит о важности создания средств поддержки самообразования.

2. Статическая модель программной системы учебного назначения

На рис. 4 представлена система основных и возможных элементов программной системы учебного назначения. Система, в свою очередь, делится на 7 обязательных и 2 возможных подсистемы. Обязательные подсистемы:

1. Управление пользователями.
2. Обучение.
3. Обслуживание.
4. Взаимодействие человек-машина.
5. Безопасность.
6. Коммуникации.
7. Учебный материал.

Возможные подсистемы:

1. Интеллект.
2. Психофизиологическое сопровождение.

Следует отметить, что представленные в данном разделе модели являются статическими, в терминах объектно-ориентированной парадигмы эти модели являются моделями классов. Ассоциации (связи между классами) представлены для облегчения понимания модели и не всегда являются полными, т.е. приемлют добавление других связей. Что касается самих элементов (классов или сущностей), они также терпимы к расширению, обобщению и детализации. Цель такого анализа состоит в выработке более доступного для понимания представления о предметной области, в которой работает класс программных систем учебного назначения.

Подсистема «Обучение» (рис. 5)

Главная задача при обеспечении решения основных педагогических задач заключается в знакомстве с основными понятиями предметной области изучаемой дисциплины, формировании навыков решения типичных для данной области задач, в проведении экспериментов с моделью (в частности со словесной моделью в случае лингвистических задач), в восстановлении и закреплении навыков и знаний, в контроле знаний.

Элементы подсистемы:

1. Задача. Необходимый элемент процесса обучения, способствующий практической деятельности обучаемого. Содержит такие элементы, как:
   (a) Тема. Название раздела дисциплины, по которой выдаётся задача.

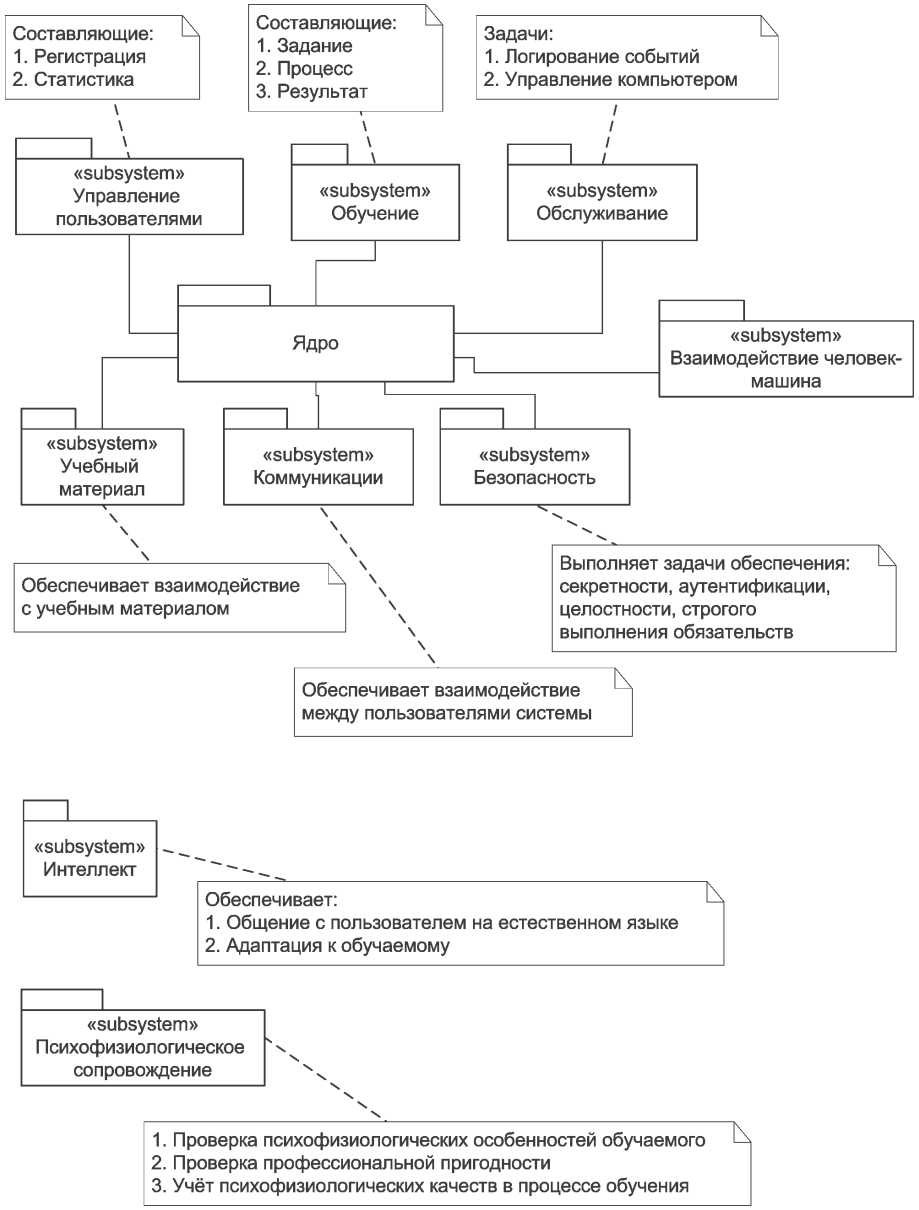

Рис. 4. Система элементов программных систем учебного назначения

(b) Условие. Содержит информацию о том, что известно и что нужно получить, в некоторых случаях есть указание на то, как нужно или можно получить решение.

(c) Ответ. Ответ к задаче.

(d) Сложность. Степень сложности задачи. Может содержать информацию об уровне сложности, указывать на степень зрелости, которой должен обладать обучаемый для успешного решения задачи.

2. Упражнение. Одна из разновидностей задач, данная обучаемому. Предполагает знакомство обучаемого с предметной областью.

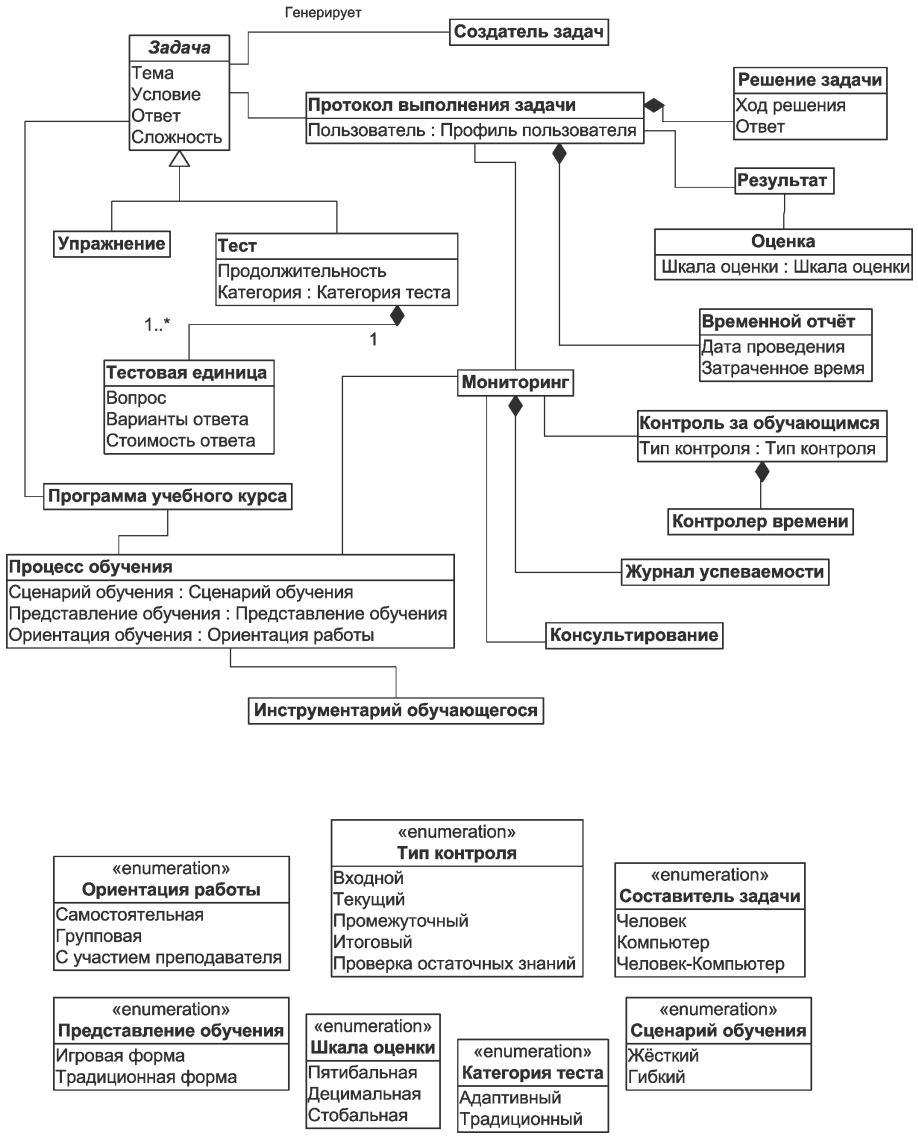

Рис. 5. Подсистема «Обучение» пятибалльная

3. Тест. Разновидность задачи, служащая для контроля знаний и умений обучаемого. Составляющие:

   (a) Продолжительность теста. Запланированная длительность теста.

   (b) Категория теста. Тип теста, виды: Адаптивный. Вопросы в таком случае подаются не в строго определённом порядке, а в зависимости от правильности или длительности ответов обучаемого на них. Традиционный. Последовательность вопросов строго определена.

4. Тестовая единица. Минимальная составляющая теста, имеющая отношение к предметной области теста. Состоит из:

   (a) Вопрос. То, на что нужно дать ответ.

(b) Варианты ответа. Возможные варианты ответов.

(c) Стоимость ответа. Цена ответа: на сколько оценивается ответ.

5. Составитель задач (Генератор задач). Элемент, позволяющий создавать задачи. Составитель задачи. Тип составителя задач, включает следующих кандидатов:

   (a) Человек. Задачи создаются вручную, благодаря интеллектуальным усилиям обучающего.

   (b) Компьютер. Задачу генерирует компьютер.

   (c) Человек-Компьютер. Человек с помощью компьютера составляет задачу.

6. Протокол выполнения задачи. Отчёт по выполненной задаче. Содержит информацию о пользователе, а именно об обучаемом.

7. Решение задачи. То, как обучаемый решил задачу. Содержит:

   (a) Ход решения. Последовательность действий решения задачи.

   (b) Ответ. Ответ, полученный обучаемым.

8. Временной отчёт. Отчёт о затраченном на работу времени. Включает:

   (a) Дата проведения. Время, когда начался процесс решения задачи.

   (b) Затраченное время. Время, когда процесс решения задачи был завершён.

9. Результат. То, что полагается пользователю за решение задачи. Одна из возможных составляющих:

   (a) Оценка. Оценивание результата работы обучаемого. Оценивание может вестись по определённой шкале. Шкала оценки (взяты из [9]): Пятибалльная. Оценки в измерении от 1 до 5. Децимальная. Оценки от 1 до 10. Стобалльная. Оценки от 1 до 100.

10. Процесс обучения (Занятие). Ход обучения. Характеризуется следующим:

    (a) Сценарий обучения. План проведения мероприятия, связанного с обучением. Виды: Жёсткий. Строго задан. Гибкий. Приспособлен к переходам в плане.

    (b) Представление обучения. То, как проходит обучение. Виды: Игровая форма. В игровой форме. Привлекаются игровые технологии. Традиционная форма. Характерная форма проведения обучения для целевой среды.

11. Ориентация работы. Направленность на количественную и качественную сторону обучения. Виды:

(a) Самостоятельная. Предполагает самостоятельную работу обучаемого.

(b) Групповая. Предполагается работа в группе обучаемых.

(c) С участием преподавателя. Предполагает работу преподавателя с одним или группой обучаемых.

12. Инструментарий обучающегося. Сущность, состоящая из специально подобранных средств для обучаемого, позволяющих эффективнее работать с материалом и выполнять учебные задачи.

13. Мониторинг. Отвечает за наблюдение работы обучаемого.

14. Консультирование. Оказание помощи обучаемому в затруднительных вопросах.

15. Журнал успеваемости. Отчёт результатов работ обучаемого с определённым курсом (дисциплиной).

16. Контроль за обучающимся. Осуществление контроля за обучаемым. Также включает контролер времени, т.е. таймер, фиксирующий и ограничивающий время работы обучающегося. Виды контроля:

    (a) Входной. Для определения уровня знаний перед изучением дисциплины.

    (b) Текущий. Используется при изучении отдельных разделов конкретной дисциплины.

    (c) Промежуточный. Определение качества усвоения отдельного раздела.

    (d) Итоговый. Проверка качества усвоения материала по целому курсу (дисциплине).

    (e) Проверка остаточных знаний. Проверка знаний, полученных обучаемым в прошлом.

Подсистема «Управление пользователями» (рис. 6)

Регистрирует пользователей в системе, составляет их профили. Ведёт статистику пользования системой.

Элементы подсистемы:

1. Регистратор пользователя. Производит регистрацию пользователя в системе.

2. Пользователь. Тот, кто регистрируется с помощью регистратора.

3. Профиль пользователя. Содержит краткую информацию о пользователе. Составляющие:

    (a) Дата регистрации. Дата первой регистрации в системе.

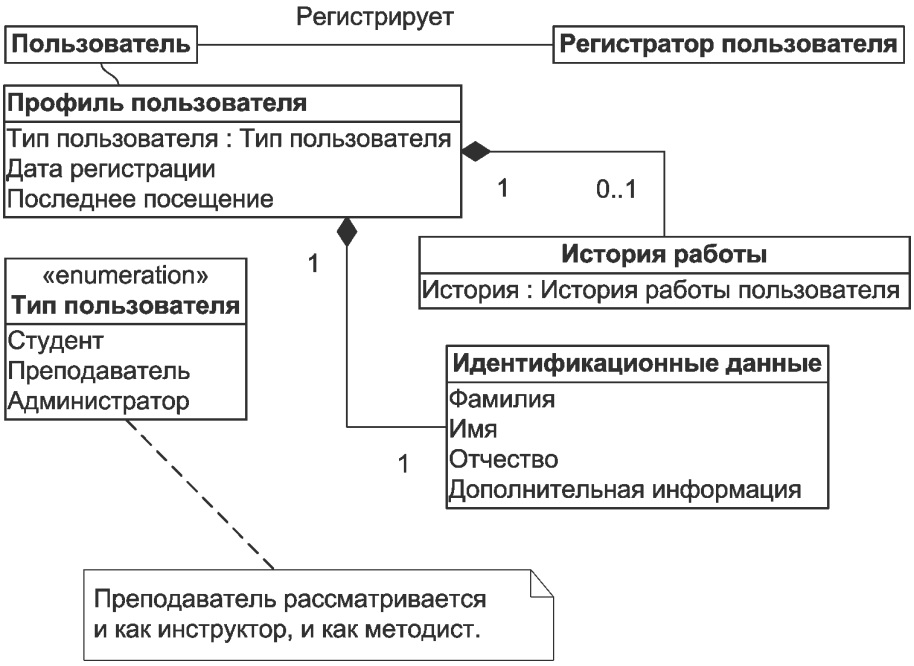

Рис. 6. Подсистема «Управление пользователями»

(b) Последнее посещение. Дата последнего посещения системы пользователем.

(c) Тип пользователя. Описание типа пользователя влияет на поведение системы в плане доступа к определённым её частям. Ограничения на доступ возможны при использовании функциональности подсистемы «Безопасность». Типы:

i. Студент. Главный пользователь системы. Тот, кто участвует в процессе обучения, на кого оно непосредственно направлено.

ii. Преподаватель. Тот, кто производит обучение пользователя типа «Студент». Составляет программу учебного курса, анализирует успеваемость обучающихся и прочее.

iii. Администратор. Управляющий системы. Отвечает за её функционирование.

4. История работы пользователя. Функционирует за счёт подсистемы «Обслуживания», производит запись информации, касающейся работы пользователя с системой: введённые команды, произошедшие события и пр. Детальность информации зависит от установленных администратором параметров.

5. Идентификационные данные. Фамилия, Имя, Отчество, Дополнительная информация (представляет собой отдельную сущность, которая может содержать адрес пользователя, телефон и прочую подобную информацию).

Подсистема «Учебный материал» (рис. 7)

Отвечает за содержание материала по предусмотренным дисциплинам, словарь терминов, и прочую информацию, касающуюся предмета. Основные функции: внесение, хранение, извлечение нужной информации с возможностью её фильтрации по различным признакам. Может содержать информацию описательного и иллюстративного характера. Возможно содержание учебных задач, сгенерированных системой или преподавателем.

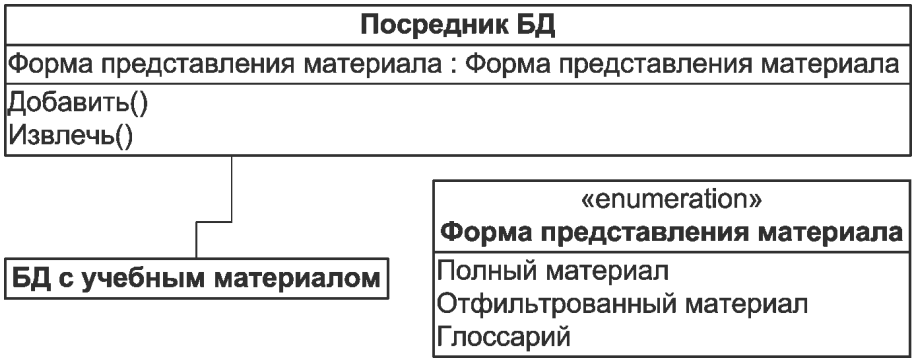

Рис. 7. Подсистема «Учебный материал»

Подсистема «Безопасность» (рис. 8)

Отвечает за такие вопросы безопасности, как: аутентификация (проверка соответствия пользователя учётной записи); секретность (иначе конфиденциальность, служит для предупреждения попадания информации пользователям, неавторизованным системой); целостность (для того чтобы быть уверенным в том, что определённые части системы не подменены и не модифицированы злоумышленником); строгое выполнение обязательств (служит для подтверждения того факта, что пользователь действительно совершил определённое действие).

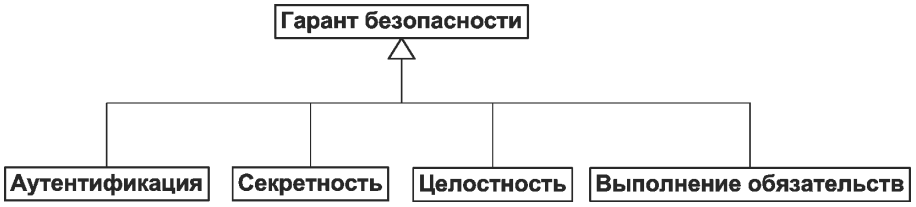

Рис. 8. Подсистема «Безопасность»

Подсистема «Коммуникации» (рис. 9)

Обеспечивает работу пользователей системы не только на локальном компьютере, но и в пределах глобальной сети.

Элементы подсистемы:

1. Коммуникация. Осуществляет взаимодействие пользователей. Тип взаимодействия пользователей:

   (а) Преподаватель-обучаемый. Предполагает работу преподавателя с обучаемым или с группой обучаемых.

(b) Обучаемый-обучаемый. Предполагает работу обучаемых друг с другом.

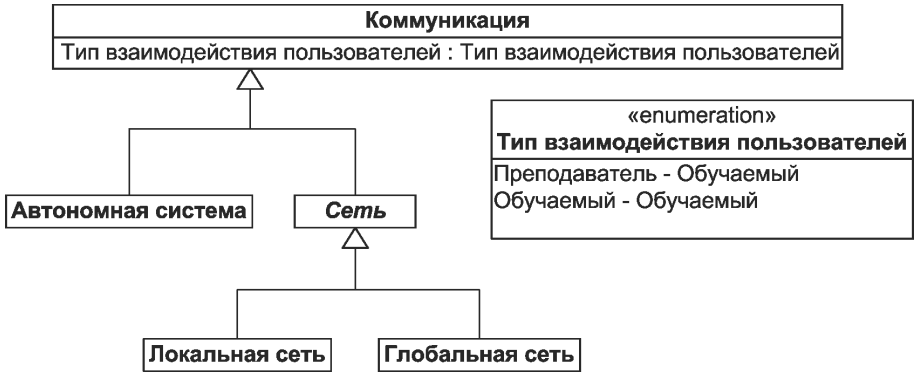

Рис. 9. Подсистема «Коммуникация»

2. Автономная система. Локальный компьютер пользователя.

3. Сеть. Совокупность компьютерных систем.

4. Локальная сеть. Объединение компьютеров в пределах предприятия, организации, учреждения.

5. Глобальная сеть. Объединение компьютеров по всему миру.

Подсистема «Обслуживание» (рис. 10)

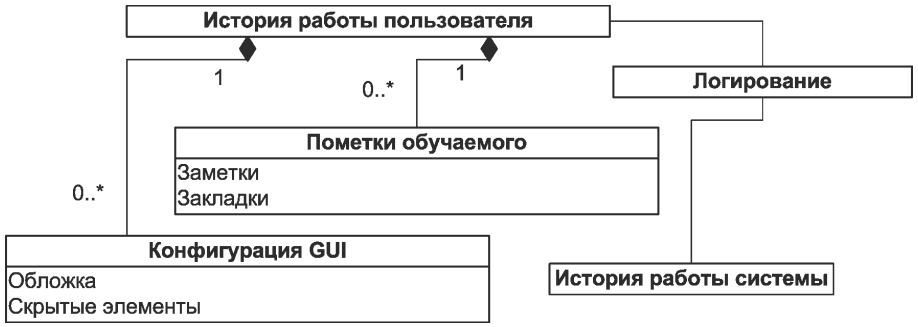

Рис. 10. Подсистема «Обслуживание»

Оказывает услуги вспомогательного характера:

1. Логирование. Фиксирование событий, происходящих в системе.

2. История работы пользователя. Содержит:

3. Пометки обучаемого. Сохранение процесса обучения на определённом этапе, закладки в учебном материале.

4. Конфигурация GUI:

(a) Обложка. Внешний вид графического интерфейса пользователя.

(b) Скрытые элементы. Элементы интерфейса, скрытые пользователем.

5. История работы системы. История работы системы, необходимая системному администратору.

Подсистема «Взаимодействие человек-машина».
Обеспечивает коммуникацию пользователя с системой.
Подсистема «Интеллект».
Предполагает общение с пользователем на естественном или условно-естественном языке, а также адаптацию к пользователю. Подсистема «Психофизиологическое сопровождение». Функции: проверка психофизиологических особенностей обучаемого, проверка профессиональной пригодности, учёт психофизиологических качеств в процессе обучения.

Заключение

В работе проведены концептуальное проектирование и анализ предметной области, в которой работают программные системы учебного назначения. Представленный материал позволяет яснее представить область, в которой и с которой работает класс рассматриваемых программных систем, даёт отправную точку для последующих стадий разработки таких систем. Следующие этапы, предполагаемые исследованиями автора, заключаются в детальном анализе приложений (подразумевается деление стадии анализа на анализ предметной области и анализ приложения) и исследовании вопросов планирования повторного использования на стадии проектирования архитектуры программных систем учебного назначения. Предполагается, что по завершении стадии анализа приложений представленная в работе модель станет более полной и в ней появятся новые элементы, связанные с особенностями приложений.

Литература

1. Яковлев, А.И. Информационно-коммуникационные технологии в образовании / А.И. Яковлев // Информационное общество. – 2001. – Вып. 2. – С. 32–37.
2. Брауде, Э. Технология разработки программного обеспечения / Э. Брауде. – СПб.: Питер, 2004.
3. Ройс, У. Управление проектами по созданию программного обеспечения. Унифицированный подход / У. Ройс. – М.: Лори, 1998.
4. Соммервилл, И. Инженерия программного обеспечения / И. Соммервилл. – 6-е изд. – М.: Вильямс, 2002.
5. Flynt, J. Software Engineering for Game Developers / J. Flynt, O. Salem. – United States of America: Thomson Course Technology PTR, 2005.
6. Башмаков, А.И. Разработка компьютерных учебников и обучающих систем / А.И. Башмаков, И.А. Башмаков. – М.: Филинъ, 2003.

7. Коротеева, О.С. Новые образовательные технологии в информационном пространстве / О.С. Коротеева, Л.В. Хорева // Образовательные технологии. – 2008. – Вып. 2. – С. 64.
8. Сибирцева, Г.А. Вопросы качества профессионального образования по заочной форме обучения / Г.А. Сибирцева, Н.Н. Самарина // Образовательные технологии. – 2008. – Вып. 2. – С. 111.
9. Нужнов, Е.В. Возможности оценки качества взаимодействия «преподаватель - обучаемый» в образовательном учреждении / Е.В. Нужнов // Перспективные информационные технологии и интеллектуальные системы. – 2005. – Вып. 3. – С. 52–55.

Математические структуры и моделирование
2009, вып. 20, с. 93–96

УДК 681.3.06

ТЕМАТИЧЕСКАЯ КЛАССИФИКАЦИЯ ДОКУМЕНТОВ ПО СТЕПЕНИ БЛИЗОСТИ ТЕРМОВ

А.С. Епрев

В статье рассматривается один из методов тематической классификации текстовых документов по близости термов, входящих в описания заданных тематик и поступающих в систему документов.

Введение

Классификация текстовых документов для электронных библиотек и баз знаний рассматривается как один из возможных вариантов решения проблемы эффективного доступа к информационным ресурсам этих систем. Проблема заключается в сложности ориентирования в этих массивах. Отсутствие возможности получить наиболее актуальную и полную информацию по конкретной теме делает бесполезной большую часть накопленных ресурсов. Использование классификаторов позволяет сократить трудозатраты на поиск нужной информации.

Тематическая классификация документов является задачей автоматического определения тематики документа по заданному множеству возможных тематик. Причем каждый документ соответствует какой-нибудь одной из заданных тематик.

Проблеме классификации текстовой информации уделено много внимания [1–3]. Большинство методов классификации основываются на использовании простой векторной модели описания документов. В рамках этой модели документ описывается вектором, в котором каждому используемому в документе терму сопоставляется его значимость в этом документе. Значимость терма рассчитывается на основе статистической информации о встречаемости термов в документе. Описание тематики также представляется вектором, и для оценки близости документа и тематики используется скалярное произведение векторов описания тематики и описания документа.

В качестве исходных данных используется матрица *термы-на-документы*. Столбцы этой матрицы – документы, а строки – термы. Элементами этой матрицы являются частоты использования данного терма в данном документе.

Copyright © 2009 **А.С. Епрев.**
Омский государственный университет им. Ф.М. Достоевского.
E-mail: a.eprev@gmail.com

Таким образом, каждый терм, документ и тематику можно представить в виде векторов в общем пространстве. Размерность этого пространства – k. Близость между любой комбинацией термов и/или документов может быть вычислена при помощи скалярного произведения этих векторов. Таким образом, чтобы отнести документ к одной из возможных тематик, достаточно вычислить наиболее близкую из возможных тематику.

1. Метод классификации документов

Классическая задача классификации документов по заданному набору тематик [2] заключается в определении для каждого поступающего в систему документа одной (или нескольких) тематик, к которым этот документ относится.

Существует множество методов классификации, но в их основе лежит один и тот же обобщенный алгоритм [3]:

- построение описаний тематик;
- построение описания поступившего в систему документа;
- вычисление степени близости описания тематик и описания документа и выбор наиболее близкой тематики.

1.1. Описания тематик и документов

Будем считать, что тематика документа определяется его «словарем». Различные синтаксические формы одного и того же слова, в словаре представляются базовой словоформой (термом). Из словаря исключаются наиболее употребительные слова, такие как предлоги, местоимения и прочие.

Описанием документа является все множество встречающихся в документе термов.

Тематики также описываются наборами термов, только эти наборы содержат не все употребляющиеся в данной тематике слова, а лишь небольшое их подмножество.

1.2. Построение описаний тематик

Тематика задается небольшим множеством относящихся к ней документов. Чтобы построить описание тематики, необходимо выявить отличия этой тематики от остальных. Это позволит сформировать набор термов, наилучшим образом подчеркивающих особенности рассматриваемой тематики.

Выбор термов для описания тематик производится при помощи следующего алгоритма:

1. Построение общего словаря термов W. В него включаются все термы, которые используются в документах, задающих тематики.

2. Для каждого терма $\omega \in W$ вычисляется оценка вероятности его использования в документах d данной тематики C:

$$P(\omega|C) = \frac{|\{d : d \in C, d \supset \omega\}|}{|C|}.$$

3. Для каждой тематики C строится тематический словарь. В этот словарь включаются термы, вероятность использования которых в рассматриваемой тематике превосходит вероятность их использования в любой другой тематике $C_i \in \Omega$, т. е.

$$P(\omega|C) \geq \frac{\sum\limits_{C_i \in \Omega} P(\omega|C_i)}{|\Omega|}.$$

Для каждого терма из тематического словаря тематики C вычисляется его значимость по следующей эмпирической формуле:

$$Order(\omega, C) = \frac{P(\omega|C)^2}{\sum\limits_{C_i \in \Omega} P(\omega|C_i)^2}.$$

4. Значимость термов $Order$ задает отношение порядка на множестве термов каждого из тематических словарей. Используя это отношение, из тематического словаря выбирается несколько наиболее значимых термов, которые будут использоваться в качестве описания этой тематики.

Оптимальное количество термов для включения в описание необходимо определить опытным путем.

1.3. Вычисление степени близости

Как уже было сказано выше, описываемый подход основывается на предположении, что тематика документа определяется его словарем.

Если мы определим функцию $Proximity(\omega_1, \omega_2)$, которая сопоставляет каждой паре термов некоторую оценку их тематической близости, то оценка тематической близости документа и тематики определяется тематической близостью термов, входящих в их описания.

Одним из вариантов оценки близости документа и тематики является среднее арифметическое попарных оценок тематической близости термов из описаний документа $d \in D$ и тематики $C \in \Omega$:

$$Hit(d, C) = \frac{\sum\limits_{\omega_i^d \in d} \sum\limits_{\omega_j^C \in C} (Proximity(\omega_i^d, \omega_j^C))}{|C| \cdot |D|}.$$

После вычисления степени близости документа с возможными тематиками можно выбрать одну или несколько тематик с наиболее высокими оценками, т. о. классифицировать документ в одну или несколько тематик.

1.4. Тематическая близость термов

Степень тематической близости пары термов характеризует, насколько часто эти термы употребляются в документах одной и той же тематики, но не обязательно присутствуют в одних и тех же документах.

Построим матрицу *термы-на-документы* X, строки которой отражают распределение термов по документам из обучающего систему набора документов. В качестве оценки тематической близости пары термов используется скалярное произведение соответствующих строк этой матрицы. Следовательно, чтобы вычислить степень тематической близости между всеми парами термов, достаточно вычислить матрицу XX^T. Таким образом, мы только что задали функцию $Proximity$:

$$Proximity(\omega_1, \omega_2) = XX^T[\omega_1, \omega_2].$$

2. Заключение

Мы рассмотрели метод классификации документов по степени близости к заданным тематикам. Однако ему присущи следующие недостатки [4, стр. 203]:

- метод не обнаруживает зависимости между термами, которые часто используются в документах одной и той же тематики, но редко встречаются вместе;
- случайные зависимости и ошибки правописания оказывают существенное влияние на получаемые оценки и негативно сказываются на точности метода;
- размер матрицы *термы-на-документы* очень велик даже для небольшого числа документов и поэтому метод весьма ресурсоемкий.

Открытым остается вопрос об оптимальном количестве термов для включения в описание тематик. Необходимо провести исследование: в рамках компьютерного эксперимента выявить зависимости качества классификации документов от числа термов, определяющих тематику.

Литература

1. Koller, D. Hierarchically classifying documents using very few words / D. Koller, M. Sahami // Proceedings of the Fourteenth International Conference on Machine Learnin. – 1997. – P. 170–178.
2. Lewis, D. A comparison of two learning algorithms for text categorization / D. Lewis, M. Ringuette // Proceedings of the 3rd Annual Symposium on Document Analysis and Information Retrieva. – 1994. – P. 81-93.
3. Кураленок, И.Е. Автоматическая классификация документов с использованием семантического анализа / И.Е. Кураленок, И.С. Некрестьянов // Программирование. – 2000. – С. 31-41.
4. Ландэ, Д.В. Основы интеграции информационных потоков / Д.В. Ландэ. – К.: Инжиниринг, 2006.

Математические структуры и моделирование
2009, вып. 20, с. 97–106

УДК 519.688

КЛАСТЕРИЗАЦИЯ МАТРИЦ С ПРОПУСКАМИ КАК МЕТОД ВОССТАНОВЛЕНИЯ МУЛЬТИМЕДИЙНОЙ ИНФОРМАЦИИ

И.Б. Ларионов

Рассматривается метод кластеризации матриц с пропусками как метод восстановления мультимедийной информации на примере графических файлов.

Введение

Надежность современных накопителей достаточно высока. Но всегда есть вероятность того, что какие-либо файлы будут испорчены. Так же современные пиринговые сети используют достаточно эффективные алгоритмы хеширования. Однако иногда происходит изменение данных, которые не заметны для внутренних алгоритмов контроля целостности носителей или для алгоритмов хеширования. Подобные изменения иногда приводят к достаточно сильному изменению данных (на примере мультимедийных данных) в связи с тем, что современные алгоритмы сжатия с потерями неустойчивы к повреждениям.

В связи с этим встает задача восстановления безвозвратно потерянной информации. Существует множество способов сглаживания потерь в мультимедийных данных с использованием интерполяции. Для случаев небольших потерь такой метод, как интерполяция кубическими сплайнами, позволяет добиться достаточно хороших результатов, что будет показано ниже. Однако при потере существенного количества информации распространенные методы интерполяции неэффективны. Такие случаи требуют более интеллектуального подхода.

Рассмотрим способ восстановления поврежденных мультимедийных данных с помощью сингулярного разложения матриц с пропусками.

1. Сингулярное разложение матриц с пропусками

Идея сингулярного разложения матриц, содержащих пропуски, на сумму одноранговых матриц была предложена старшим научным сотрудником Новосибирского института математики С.В. Макаровым. И хотя материал этого раздела

Copyright © 2009 **И.Б. Ларионов.**
Омский Государственный Университет им.Ф.М.Достоевского.
E-mail: igor@1.ws

непосредственно в обработке данных не используется, он дает нам простейший пример и прототип для дальнейших построений.

Пусть задана прямоугольная матрица $A = (a_{ij})$, клетки которой заполнены действительными числами или значком @, означающим отсутствие данных. Требуется представить исходную матрицу A в виде суммы одноранговых матриц P_q:

$$A = \sum_q P_q.$$

Таким образом, ставится задача поиска наилучшего приближения A матрицей вида $x_i y_j$ методом наименьших квадратов:

$$\Phi = \sum_{\substack{i,j \\ a_{ij} \neq @}} (a_{ij} - x_i y_j)^2 \to \min. \tag{1}$$

Решение этой задачи дается последовательными итерациями по явным формулам. При фиксированном векторе y_i значения x_i, доставляющие минимум форме (1), однозначно и просто определяются из равенств $\delta\Phi/\delta x_i$:

$$\frac{\delta\Phi}{\delta x_i} = -2 \sum_{\substack{j \\ a_{ij} \neq @}} (a_{ij} - x_i y_j) y_j = 0,$$

$$x_i = \Big(\sum_{\substack{j \\ a_{ij} \neq @}} a_{ij} y_j \Big) \Big/ \Big(\sum_{\substack{j \\ a_{ij} \neq @}} (y_j)^2 \Big).$$

Введем обобщенное на случай данных с пробелами определение скалярного произведения $(\bullet, \bullet)_a$ и нормы $|| \bullet ||_a$.

Определение 1. Скалярное произведение $(y_1, y_2)_a$ векторов y_1 и y_2 называется скалярным произведением по известным компонентам вектора a и считается следующим образом:

$$(y_1, y_2)_a = \sum_{\substack{i \\ a_i \neq @}} y_{1i} y_{2i}.$$

Определение 2. Норма $||y||_a$ вектора y называется нормой по известным компонентам вектора a и считается следующим образом:

$$||y||_a = \sqrt{(y, y)_a} = \sqrt{\sum_{\substack{i \\ a_i \neq @}} y_i^2}.$$

С учетом этих определений можно заметить, что значения проекций x_i находятся как нормированное скалярное произведение вектора данных a_i (i-ая строка матрицы A) на вектор y:

$$x_i = \frac{(a_i, y)_{a_i}}{||y||_{a_i}^2},$$

где, напомним, скалярное произведение $(\bullet,\bullet)_{a_i}$ и норма $||\bullet||_{a_i}$ вычисляются по известным компонентам вектора a_i, т.е. мы имеем дело с обобщенным на случай данных с пробелами скалярным произведением и нормой.

Аналогично и при фиксированном векторе x_i значение y_i, доставляющее минимум форме (1), определяется явно из равенств $\delta\Phi/\delta y_j = 0$:

$$\frac{\delta\Phi}{\delta y_j} = -2 \sum_{\substack{i \\ a_{ij}\neq @}} (a_{ij} - x_i y_j) x_i = 0,$$

$$y_i = \Big(\sum_{\substack{i \\ a_{ij}\neq @}} a_{ij} x_i \Big) \Big/ \Big(\sum_{\substack{i \\ a_{ij}\neq @}} (x_i)^2 \Big).$$

Аналогично вычислению проекции x_i через скалярное произведение можно записать соответствующую проекцию j-го столбца a_j матрицы A на вектор x:

$$y_j = \frac{(a_j, x)_{a_j}}{||x||_{a_j}{}^2},$$

где скалярное произведение и норма вычисляются по известным компонентам вектора a_j. Процесс вычисления итерационный, поэтому в качестве начального приближения вектора y возьмем случайное значение, но потребуем, чтобы y был единичной длины:

$$y - \text{случайный, нормирован на 1 (т.е.} ||y||^2 = \sum_j y_j{}^2 = 1).$$

В качестве критерия остановки будем использовать малость относительного улучшения значения минимизируемого функционала на итерации, т.е. критерий остановки малость относительного улучшения $\Delta\Phi/\phi$, где $\Delta\Phi$ – полученное за цикл уменьшение значения Φ, а Φ – само текущее значение. Естественно и использование второго критерия остановки - малость самого значения Φ.

Таким образом, итерационная процедура останавливается, если $\Delta\Phi/\Phi < \epsilon$ или $\Phi < \delta$ для некоторых ϵ, $\delta < 0$.

В результате для матрицы A получили наилучшее приближение матрицей P_1 вида $x_i y_j$. Далее, из матрицы A вычитаем полученную матрицу P_1, и для полученной матрицы уклонений $A - P_1$ вновь ищем наилучшее приближение P_2 этого же вида и т.д., пока, например, норма A не приблизится в достаточной степени к нолю.

В результате получили опять же итерационную процедуру разложения матрицы A в виде суммы матриц ранга 1, т.е. $A = P_1 + P_2 + \cdots + P_q$.

Из теории сингулярного разложения матрицы в виде суммы одноранговых матриц известно, что в случае полной (без пробелов) матрицы число полученных одноранговых матриц не превышает число столбцов исходной матрицы. В общем же случае при наличии пробелов это не так.

2. Описание наиболее распространенных повреждений графических файлов

В ходе эксперимента были предприняты попытки восстановить графические файлы формата BMP. Этот формат был выбран как наиболее простой и как один из форматов, в котором отсутствует сжатие, при котором изменение одного бита повлечет за собой изменение всех данных, идущих после этого бита (как это происходит в алгоритме JPEG). Также этот формат наиболее похож на формат RAW, который используют все профессиональные фотографы. Формат RAW отличается от BMP только заголовком, что позволяет рассматривать два этих формата как один и тот же. При стандартной потере одного блока (будь то блок на жестком диске или пакет в сети) данный формат теряет горизонтальный блок точек (пикселей) шириной в один пиксель. Это вызвано тем, что все 3 (а иногда и 4, если используется alpha-канал) канала изображения идут последовательно друг за другом, сгруппированные в блок из 3 (4) байт для одного пикселя.

В рассматриваемом формате данные хранятся в формате (a)RGB, т.е. хранятся отдельные составляющие для красного, зеленого и синего цветов (так же прозрачность alpha-канала). При моделировании восстановления данных было принято решение об использовании цветовой модели YUV, т.к. эта модель наиболее распространена (современное аналоговое телевидение, MPEG-подобные кодеки).

В модели YUV есть три составляющие: Y - яркость и две цветоразностных - U и V. Преобразование из RGB в YUV происходит по следующим формулам:

$$Y = 0.299R + 0.587G + 0.114B,$$

$$U = -0.14713R - 0.28886G + 0.436B,$$

$$V = 0.615R - 0.51499G - 0.10001B.$$

Из YUV в RGB:

$$R = Y + 1.13983V,$$

$$G = Y - 0.39465U - 0.58060,$$

$$B = Y + 2.03211U.$$

Как видно из этих формул, данная цветовая модель позволяет проводить эксперименты по восстановлению, используя только одну составляющую - яркость, которая в свою очередь содержит информацию о всех трех цветовых составляющих. Результатов данных испытаний вполне достаточно для сравнения результатов метода восстановления с результатами распространенных методов интерполяции.

В качестве эталонного метода интерполяции был выбран метод бикубических сплайнов, как наиболее ресурсоёмкий и наиболее распространенный при обработке изображений.

3. Интерполяция бикубическими сплайнами

Пусть на отрезке $a \le \xi \le b$ задана сетка $\omega = \{x_i : x_0 = a < x_1 < \cdots < x_i < x_n = b\}$ и в её узлах заданы значения функции $y(x)$, равные $y(x_0) = y_0, \cdots, y(x_i) = y_i, \cdots, y(x_n) = y_n$. Требуется построить интерполянту — функцию $f(x)$, совпадающую с функцией $y(x)$ в узлах сетки:

$$f(x_i) = y_i, i = 0, 1, \cdots, n. \tag{2}$$

Основная цель интерполяции — получить быстрый (экономичный) алгоритм вычисления значений $f(x)$ для значений x, не содержащихся в таблице данных.

Интерполируюшие функции $f(x)$, как правило, строятся в виде линейных комбинаций некоторых элементарных функций:

$$f(x) = \sum_{k=0}^{N} c_k \Phi_k(x_i),$$

где $\{\Phi_k(x)\}$ — фиксированный линейно независимые функции, $c_0, c_1, \cdots, c_n$ — не определенные пока коэффициенты. Из условия (2) получаем систему из $n+1$ уравнений относительно коэффициентов $\{c_k\}$:

$$\sum_{k=0}^{N} c_k \Phi_k(x_i) = y_i, i = 0, 1, \cdots, n.$$

Предположим, что система функций $\Phi_k(x)$ такова, что при любом выборе узлов $a < x_1 < \cdots < x_i < \cdots < x_n = b$ отличен от нуля определитель системы:

$$\Delta(\Phi) = \begin{vmatrix} \Phi_0(x_0) & \Phi_1(x_0) & \cdots & \Phi_n(x_0) \\ \Phi_0(x_1) & \Phi_1(x_1) & \cdots & \Phi_n(x_1) \\ \cdots & \cdots & \cdots & \cdots \\ \Phi_0(x_n) & \Phi_1(x_n) & \cdots & \Phi_n(x_n) \end{vmatrix}.$$

Тогда по заданным $y_i (i = 1, \cdots, n)$ однозначно определяются коэффициенты $c_k (k = 1, \cdots, n)$.

Интерполяция кубическими сплайнами является частным случаем кусочно-полиномиальной интерполяции. В этом специальном случае между любыми двумя соседними узлами функция интерполируется кубическим полиномом, его коэффициенты на каждом интервале определяются из условий сопряжения в узлах:

$$f_i = y_i, f'(x_i - 0) = f'(x_i + 0), f''(x_i - 0) = f''(x_i + 0), i = 1, 2, \cdots, n - 1.$$

Кроме того, на границе при $x = x_0$ и $x = x_n$ ставятся условия

$$f''(x_0) = 0, f''(x_n) = 0. \tag{3}$$

Будем искать кубический полином в виде

$$f(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3, x_{i-1} \le \xi \le \xi_i. \tag{4}$$

Из условия $f_i = y_i$ имеем

$$f(x_{i-1}) = a_i = y_{i-1}, f(x_i) = a_i + b_i h_i + c_i h_i^2 + d_i h_i^3 = y_i, h_i = x_i - x_{i-1}, i = 1, 2, \cdots, n-1. \tag{5}$$

Вычислим производные:

$$f'(x) = b_i + 2c_i(x - x_{i-1}) + 3d_i(x - x_{i-1})^2, f''(x) = 2c_i + 6d_i(x - x_{i-1}), x_{i-1} \le \xi \le x_i,$$

и потребуем их непрерывности при $x = x_i$:

$$b_{i+1} = b_i + 2c_i h_i + 3d_i h_i^2, c_{i+1} = c_i + 3d_i h_i, i = 1, 2, \cdots, n-1. \tag{6}$$

Общее число неизвестных коэффициентов, очевидно, равно $4n$, число уравнений (5) и (6) равно $4n-2$. Недостающие два уравнения получаем из условия (3) при $x = x_0$ и $x = x_n$:

$$c_1 = 0, c_n + 3d_n h_n = 0.$$

Выражение из (6) $d_i = \frac{c_{i+1} - c_i}{3h_i}$; подставляя это выражение в (5) и исключая $a_i = y_{i-1}$, получим

$$b_i = \left[\frac{y_i - y_{i-1}}{h_i}\right] - \frac{1}{3}h_i(c_{i+1} + 2c_i), i = 1, 2, \cdots, n-1, b_n = \left[\frac{y_n - y_{n-1}}{h_n}\right] - \frac{2}{3}h_n c_n. \tag{7}$$

Подставив теперь выражения для b_i, b_{i+1} и d_i в первую формулу (6), после несложных преобразований получаем для определения c_i разностное уравнение второго порядка

$$h_i c_i + 2(h_i + h_{i+1})c_{i+1} + h_{i+1}c_{i+2} = 3\left(\frac{y_{i+1} - y_i}{h_{i+1}} - \frac{y_{i+1} - y_i}{h_{i+1}}\right), i = 1, 2, \cdots, n-1. \tag{8}$$

С краевыми условиями

$$c_1 = 0, c_{n+1} = 0. \tag{9}$$

Условие $c_{n+1} = 0$ эквивалентно условию $c_n + 3d_n h_n = 0$ и уравнению $c_{i+1} = c_i + d_i h_i$. Разностное уравнение (8) с условиями (9) можно решить методом прогонки, представив в виде системы линейных алгебраических уравнений вида $A * x = F$, где вектор x соответствует вектору $\{c_i\}$, вектор F поэлементно равен правой части уравнения (8), а матрица A имеет следующий вид:

$$A = \begin{pmatrix} C_1 & B_1 & 0 & 0 & \cdots & 0 & 0 \\ A_2 & C_2 & B_2 & 0 & \cdots & 0 & 0 \\ 0 & A_3 & C_3 & B_3 & \cdots & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & B_{n-1} \\ 0 & 0 & 0 & 0 & \cdots & A_n & C_n \end{pmatrix}, \tag{10}$$

где $A_i = h_i, i = 2, \cdots, n, B_i = h_{i+1}, i = 1, \cdots, n-1$ и $C_i = 2(h_i + h_{i+1}), i = 1, \cdots, n$.

4. Метрики сравнений изображений

В ходе работы встал вопрос о методе сравнения качества восстановления изображений. Были выбраны насколько метрик, основанные на разнице пикселей.

4.1. Метрика Минковского (MM)

Норма разницы между изображениями может быть посчитана путем взятия Минковского среднего разниц пикселей сначала пространственно, а затем хроматически (т.е. по зонам):

$$\epsilon^{\gamma} = \frac{1}{K}\sum_{k=1}^{K}\left\{\frac{1}{N^2}\sum_{i,j=0}^{N-1}\left|C_k(i,j) - \hat{C}_k(i,j)\right|^{\gamma}\right\}^{1/\gamma},$$

где $\hat{C}_k(i,j)$ - значение цветосоставляющей пикселя восстановленного изображения.

Для значения $\gamma = 2$ мы получаем формулу для вычесления среднеквадратической ошибки, которая описана ниже.

В ходе работы использовалось значение $\gamma = \infty$:

$$\epsilon^{\infty} = \max_{i,j}\sum_{k=1}^{K}\frac{1}{K}\left|C_k(i,j) - \hat{C}_k(i,j)\right| = \max_{i,j}||\mathbf{C}(i,j) - \hat{\mathbf{C}}(i,j)||.$$

4.2. Среднеквадратическая ошибка (MSE)

Данная метрика является частным случаем метрики Минковского:

$$\epsilon = \frac{1}{N^2}\sum_{i,j=0}^{N-1}\left(C(i,j) - \hat{C}(i,j)\right)^2.$$

4.3. Метрика разницы с соседями (DON)

Данная метрика толерантна к сдвигам пикселей и обычно используется в оценке качества сжатия видеоряда.

$$\epsilon = \sqrt{\frac{1}{2(N-w)^2}\sum_{i,j=w/2}^{N-w/2}\left[A^2 + B^2\right]},$$

$$A = \min_{l,m\in w_{i,j}}\left\{d(\mathbf{C}(i,j), \hat{\mathbf{C}}(i,m))\right\},$$

$$B = \min_{l,m\in w_{i,j}}\left\{d(\hat{\mathbf{C}}(i,j), \mathbf{C}(l,m))\right\},$$

где $d(\bullet,\bullet)$ – некая функция расстояния. В данной работе рассматривается Евклидово расстояние.

Нетрудно заметить, что данная метрика при $w = 1$ сводится к среднеквадратической ошибке, описанной выше.

4.4. Метрика многомерного расстояния (MDM)

Данная метрика основывается на том, что большинство изображений хранятся в больших разрешениях (2000 пикселей в каждом измерении и более):

$$\epsilon = \sum_{r=1}^{R} \left(\frac{1}{2^r} \frac{1}{2^{2r-3}} \sum_{i,j=1}^{2^{r-1}} \left[(g_{ij}^R - \hat{g}_{ij}^R)^2 + (g_{ij}^B - \hat{g}_{ij}^B)^2 + (g_{ij}^G - \hat{g}_{ij}^G)^2 \right]^{1/2} \right),$$

где, для примера, g_{ij}^R - среднее значение серой составляющей ij-того блока красной составляющей, а r - относительное разрешение метрики.

5. Сравнение методов восстановления

В ходе работы было принято правило, что существует некая карта утраченных пикселов. Это было сделано для упрощения алгоритмов. На практике определение таковых потерь не является сложным, однако в контексте рассматриваемой темы, данный вопрос не является критичным.

Были реализованы два метода восстановления потерь и 4 метода сравнения восстановленных изображений с эталонным. Для упрощения вычислений метрики вычислялись только по карте потерь.

Было проведено 100 испытаний на каждом из 20 изображений различных типов:

1. Высококачественные фотографии NASA.
2. Высококачественные фотографии пейзажей.
3. Фотографии групп людей.
4. Различные искусственные изображения - логотипы, векторная графика, фрактальные изображения, которые были преобразованы в формат BMP.

Данные типы изображений были выбраны из соображений наиболее полного охвата возможных входных наборов данных.

Для каждого испытания выбирался случайный потерянный блок размером в 64Кб. Причем, начало блока не было кратно ни 64Кб (как номер блока от начала файла), ни 4 байт (как номер потерянного пикселя относительно начала). После выбора границ блока составлялась карта потерянных данных, основываясь на структуре файла и на тех пикселях, любая составляющая которых входила в потерянный блок. Для приближения к реальным условиям пиксель, в котором хотя бы одна составляющая потеряна, признавался потерянным не полностью и обрабатывался отдельно, без помощи цветовой модели YUV.

Результаты, полученные в ходе испытаний, представлены в таблице 1 (результат в отдельной группе взят как среднее арифметическое всех испытаний этой группы).

Таблица разбита на 4 группы по 2 столбца в каждой. Каждая группа отображает результаты соответствующей метрики: MM – метрика Минковского; MSE

Таблица 1. Результаты испытаний

Метрика	MM		MSE		DON		MDM	
Метод	Класт.	Спл.	Класт.	Спл.	Класт.	Спл.	Класт.	Спл.
Высококачественные фотографии NASA	246,51	12,14	6345,20	276,58	876,45	23,48	758,45	90,23
Высококачественные фотографии пейзажей	645,02	34,58	9437,69	234,76	589,32	64,75	923,47	65,89
Фотографии групп людей	234,76	59,23	4753,29	347,60	289,34	75,98	234,76	59,82
Искусственные изображения	96,45	88,23	87,65	62,37	85,23	56,56	96,27	98,43

– среднеквадратическая ошибка; DON – метрика разницы с соседями; MDM – метрика многомерного расстояния.

Из таблицы с результатами видно, что кластеризация данных с пропусками по всем статьям проигрывает интерполяции бикубическими сплайнами, кроме случаев искусственных изображений. Это можно объяснить тем, что в искусственных изображениях доля линейных составляющих велика, в отличие от изображений реального мира.

Выводы

В ходе работы были реализованы два метода восстановления потерянной графической информации – кластеризация матриц с пропусками (линейная одномерная модель) и интерполяция бикубическими сплайнами. Было проведено сравнение этих методов метриками оценки качества восстановления, основанных на разнице значений пикселов.

Результаты испытания методов показали, что кластеризация матриц с пропусками не является достаточно качественным методом для восстановления фотографий. Однако для восстановления изображений, созданных искусственно, данный метод вполне подходит и показывает результаты восстановления, не уступающие в достаточной мере методу интерполяции бикубическими сплайнами.

Результаты показали, что метод восстановления изображений, основанный на кластеризации матриц, имеет право на жизнь. Однако, для достаточно точного восстановления фотографий, необходимо применять модель, отличную от линейной одномерной.

Литература

1. Горбань, А.Н. Итерационный метод главных компонент для таблиц с пробелами / А.Н. Горбань, С.В. Макаров, А.А. Россиев // Третий сибирский конгресс по прикладной и индустриальной математике (ИНПРИМ-98), 22-27 июня 1998. – Тезисы докладов. Ч. 5. – Новосибирск: Изд-во Института математики СО РАН, 1998.
2. Самарский, А.А. Введение в численные методы / А.А. Самарский. – М.: Наука, 1982.
3. Avcibas, I. Image Quality Statistics and Their Use in Steganalysis and Compression / Ismail Avcibas, Bulent Sankur, Lale Akarun, Emin Anarim, Nasir Memon, Yucel Yemez. – 2001.

Математические структуры и моделирование
2009, вып. 20, с. 107–112

УДК 004.492.3

МЕТОД КЛАССИФИКАЦИИ ГИСТОГРАММ ДЛЯ ФИЛЬТРАЦИИ СПАМ-ИЗОБРАЖЕНИЙ

Д.А. Лыфарь, В.В. Коробицын

A method for fast classification of spam images based on using machine learning AdaBoost classifier for color and grayscale image histograms is presented. False positives and false negatives are evaluated. Suggestions about using the method as a filter in real antispam engine are given.

1. Введение

Современные методы фильтрации спама основываются не только на анализе содержимого тела письма, поскольку методы рассылки спама совершенствуются. Даже человек по содержанию письма не сразу способен отличить спам-сообщения от обычных писем. Так, если раньше простейший фильтр Байеса, предложенный в 2002 году Полом Грэмом [1], позволял существенно снизить объем спама с очень низким количеством позитивных промахов, то в настоящее время этого недостаточно. Существует множество способов для спам-сообщений обойти фильтрацию фильтром, работающим на основе анализа текстового содержимого письма. К их числу относятся: рассылка спам-сообщений с графическими вложениями; сообщений, тело которых содержит не рекламный текст, а представляет собой обычное письмо с ссылками на рекламные сайты. В последнее время участились случаи взлома злоумышленниками сайтов, IP адрес которых имеет хорошую репутацию, и помещении страниц на взломанный сайт, в которых содержится перенаправление (redirect) на рекламный. Далее в спам-сообщение вставляется подобная ссылка, которая не проходит ни фильтрацию по содержимому, ни фильтрацию по IP/URL. Выявление подобных атак возможно сводится к постобработке данных письма и обучении антиспам-базы.

В данной статье рассмотрен вид графических спам сообщений и предложен метод их фильтрации.

2. Характерные признаки спам-изображений

Характерным примером графического спама может служить изображение, приведенное на рисунке 1а. Для фильтрации подобных изображений обычно приме-

Copyright © 2009 **Д.А. Лыфарь, В.В. Коробицын.**
Омский государственный университет.
E-mail: dlyfar@gmail.com

няется метод оптического распознавания символов. Примером применения подобного метода фильтрации может служить известный сервис почты gmail [2]. Однако оптическое распознавание требует много вычислительных ресурсов и становится неэффективным, когда изображение пропускается через ряд графических фильтров с целью искажения. Так же подобные фильтры обычно натренированы на распознавание ограниченного множества языков [3].

а) б)

Рис. 1. Изображения: а) спам, б) обычное

Предлагаемый метод работает вне зависимости от языка или степени искаженности текста. Стоит заметить, что этот метод фильтрации может быть использован в качестве основной оценки, однако чтобы снизить число изображений, которые распознаны неверно — необходимо подкрепить результат фильтра рядом других признаков, говорящих о принадлежности письма к классу спам-сообщений (например, фильтрация по IP из заголовка письма).

Заметим, что в отличие от нормального изображения в спам-изображении большую часть занимает текст. Если построить grayscale гистограмму для обычного и спам-изображений станет очевидным их различие по распределенности компонент на гистограмме (при построении гистограмм использовалось число корзин B=64). На рисунке 2 заметно, что у типичного спам-изображения присутствует несколько пиков, в то время как у обычного изображения компоненты гистограммы распределены равномерно (см. рис. 3). Это один из признаков, который будет учитываться в принятии решении о классификации данного изображения.

Фильтрация на основании данных grayscale гистограммы особенно эффективна, когда спам-изображение содержит в себе в основном текст. Те спам-изображения, в которых присутствует не только текст, лучше поддаются классификации на основании данных цветовых гистограмм (в RG или HV-пространстве). Мы считаем изображение спамом, когда оба классификатора имеют этот результат, чтобы уменьшить число позитивных промахов. Позитивным промахом (false positive) принято считать ситуацию, в которой классификатор ошибочно считает нормальное письмо спамом, негативным промахом (false negative) принято считать классификацию спама как нормального изображения.

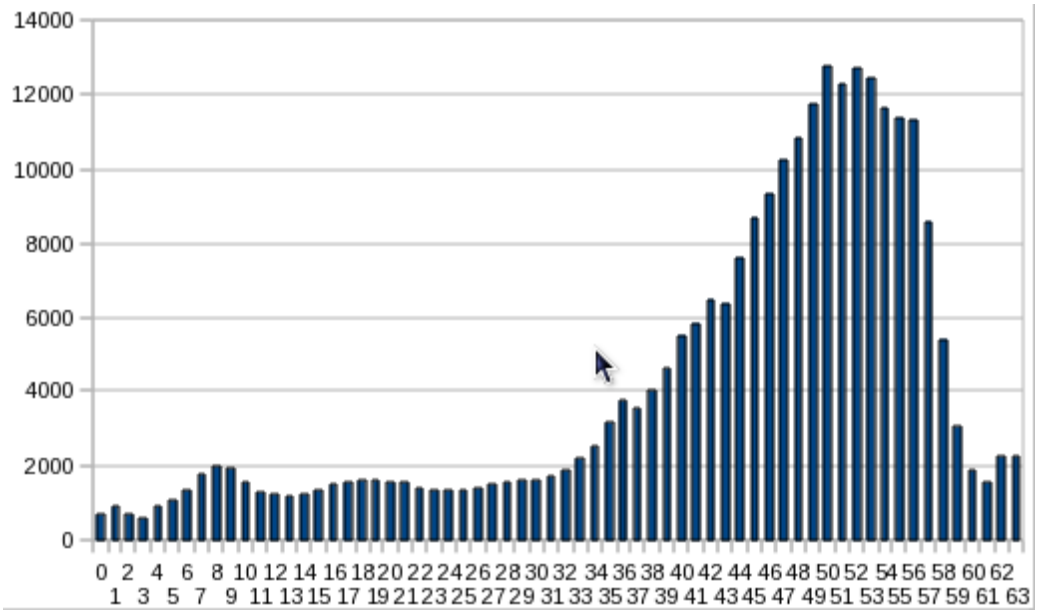

Рис. 2. Grayscale гистограмма для спам-изображения

3. Классификация изображений

В этой работе мы использовали классификатор AdaBoost. Сначала мы должны обучить классификатор уже известным нормальным и спам-изображениям, чтобы сформировать базу данных изображений, на основании которой будут делаться предположения о классе данного изображения. Алгоритм работы AdaBoost подробно описан в [4], здесь мы приводим лишь краткое описание. Этот алгоритм был успешно использован во многих областях, в частности для задачи поиска лиц на изображении.

Требуется построить классифицирующую функцию $F : X \to Y$, где X - пространство векторов признаков (в нашем случае это данные grayscale и цветовых гистограмм), Y - пространство меток классов (в нашем случае это два класса: спам- и нормальное изображение). Пусть в нашем распоряжении имеется обучающая выборка $(x_1, y_1), ..., (x_n, y_n)$, где $x_i \in X$ — вектор признаков, а $y_i \in Y$ — метка класса, к которому принадлежит x_i. Далее в статье мы будем рассматривать задачу с двумя классами, то есть $Y = \{-1; +1\}$. Также у нас есть семейство простых классифицирующий функций $H : X \to Y$. Мы будем строить финальный классификатор в следующей форме:

$$F(x) = \sum_{m=0}^{M} \alpha_m h_m(x). \tag{1}$$

Построим итеративный процесс, где на каждом шаге будем добавлять новое слагаемое

$$f_m = \alpha_m h_m(x), \tag{2}$$

вычисляя его с учётом работы построенной части классификатора (f_0, f_1,..., f_{n-1}). Приведем псевдокод алгоритма AdaBoost:

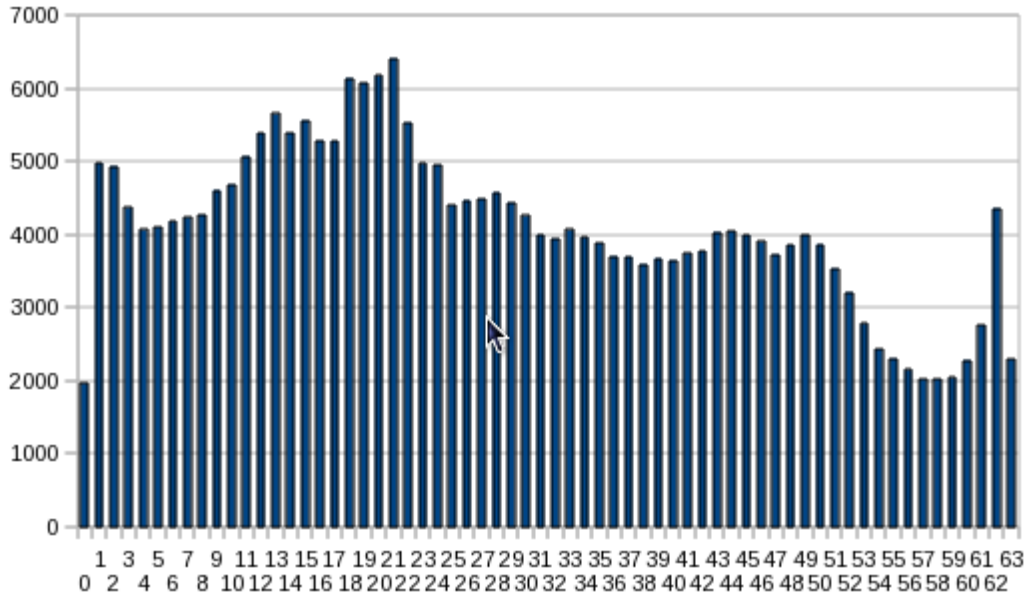

Рис. 3. Grayscale гистограмма для обычного изображения

1. Пусть задана обучающая выборка $(x_1, y_1), \ldots, (x_N, y_N)$ и распределение весов $D_1(i) = 1/N$.

2. Для каждого шага $m = 1, 2, \ldots, M$ выполнить:

   а) выбрать наилучший для текущего распределения $D_m(i)$ слабый классификатор $h_m(x) \in H$ по формуле

   $$h_m = \arg\min_{h_j \in H} \epsilon_j = \sum_{i=1}^{N} D_m(i)[y_i \neq h_j(x_i)];$$

   б) вычислить коэффициент $\alpha_m = \frac{1}{2}\log(\frac{1-\epsilon_m}{\epsilon_m})$;

   в) запомнить $f_m(x) = \alpha_m h_m(x)$ и обновить распределение

   $$D_{m+1}(i) = \frac{D_m(i)\exp(-\alpha_m y_i h_m(x_i))}{Z_m},$$

   где Z_m — нормирующий коэффициент, обеспечивающий выполнение условия $\sum\limits_{i=1}^{N} D_{m+1}(i) = 1$.

3. Составляем итоговый классификатор:

   $$F(x) = \operatorname{sgn}\left[\sum_{m=1}^{M} f_m(x)\right].$$

4. Результаты эксперимента

Мы использовали реализацию алгоритма AdaBoost из открытой библиотеки opencv. В качестве набора для обучения было использовано 946 нормальных изображений и 825 спам-изображений. Полученный классификатор работал на реальном потоке электронных сообщений одного из провайдеров Европы, результаты потока помогли составить приблизительную оценку эффективности этого метода. Как уже было сказано выше, под эффективностью метода фильтрации понимается процент позитивных (FP) и негативных (FN) промахов.

Эксперименты показали, что эффективность предсказания зависит от числа корзин (B=64 — это эмпирически подобранное значение, с дальнейшим ростом B эффективность метода практически не изменялась), от размера изображения (все изображения приводились к размеру 512x512 пикселей). Результаты показали $FP = 0.014$ и $FN = 0.12$ для 100 изображений из потока. При этом были выявлены следующие преимущества и недостатки метода.

Преимуществами можно считать высокую производительность и низкий процент FP. Производительность метода значительно выше, чем у методов, использующих алгоритмы распознавания текста. Достаточно низкий процент FP позволяет использовать этом метод в качестве вспомогательного фильтра и в качестве основного во время спам-атак для писем, содержащих только изображения и пришедших с белых IP-адресов.

К недостаткам можно отнести значительно высокое FN. Фильтр ошибочно определяет некоторый набор изображений как спам: сканированные документы, изображения из новостных рассылок. Фильтр слабо реагирует на те рекламные изображения, где большую площадь занимает изображение, идентичное нормальному.

5. Заключение

Представленный метод фильтрации писем целесообразно использовать в цепочке фильтров антиспам-системы для фильтрации спам-изображений. Так как сегодня спам-атаки, содержащие изображения, время от времени представляют собой достаточно высокий процент писем, данный фильтр является оптимальным выбором с точки зрения производительности и качества. Чтобы уменьшить число позитивных промахов, целесообразно будет использовать фильтр в качестве дополнительного при оптическом распознавании символов в той ситуации, когда фильтр классифицирует сообщение как спам.

Литература

1. Graham, P. A Plan for Spam [Электронный ресурс] / P. Graham. – Режим доступа: http://www.paulgraham.com/spam.html (2.10.2009).
2. Gmail uses Google's innovative technology to keep spam out of your inbox [Электронный ресурс] – Режим доступа: http://www.google.com/mail/help/fightspam/spamexplained.html (3.10.2009).

3. Tesseract-ocr - Project Hosting on Google Code [Электронный ресурс] – Режим доступа: http://code.google.com/p/tesseract-ocr/ (3.10.2009).
4. Sochman, J., Matas, J. AdaBoost [Электронный ресурс] / J. Sochman, J. Matas. – Режим доступа: http://cmp.felk.cvut.cz/ sochmj1/adaboost_talk.pdf (3.10.2009).
5. Херн, Д., Бейкер, М.П. Компьютерная графика и стандарт OpenGL. – М.: Издательский дом «Вильямс», 2005.

Математические структуры и моделирование
2009, вып. 20, с. 113–120

УДК 004.451:004.514

НАСТРОЙКА XKB НА ТОНКИХ КЛИЕНТАХ SUN RAY

А.А. Печерицын

В данной работе описывается механизм работы модуля XKB, управляющего вводом с клавиатуры в графической оболочке X Window System. Предложено решение проблем, возникающих при настройке XKB на тонких клиентах Sun Ray 2.

Введение

В современных версиях X Window System (графической подсистемы Unix-подобных ОС) вводом с клавиатуры управляет модуль X Keyboard Extension (XKB). Примеры установки нескольких раскладок клавиатуры и переключения между ними через XKB можно найти в руководствах по ОС Unix и в сети Internet. Однако если настройка XKB не дает ожидаемого результата, то готовые способы решения этой проблемы в Internet найти, как правило, сложно.

В данной работе приведено краткое изложение основ XKB и показано, как, используя утилиты для работы с XKB, получить дополнительную информацию о текущих параметрах ввода с клавиатуры. Далее показано, как применить полученные знания к решению проблем настройки XKB на тонких клиентах Sun Ray 2.

1. Настройки XKB

В этом разделе приводятся необходимые для дальнейшего изложения сведения о механизме работы и настройках XKB, взятые из руководства [1].

Основное назначение модуля XKB — преобразование скан-кодов нажимаемых клавиш в коды символов. Для этого XKB использует таблицу символов, в которой с каждой клавишей (точнее, скан-кодом) связано несколько однострочных таблиц символов (XKB-групп). Каждая такая таблица делится на несколько колонок. В колонках находятся символы, которые может выдавать данная клавиша. При нажатии клавиши сначала по значению специальной переменной — текущему номеру группы — определяется, какую группу нужно использовать. Затем по состоянию клавиатурных модификаторов (Shift, Ctrl и

Copyright © 2009 **А.А. Печерицын.**
Омский государственный университет.
E-mail: pecher@cmm.univer.omsk.su

т.д.) выбирается колонка, откуда будет взято выдаваемое значение. Количество групп, связанных с данной клавишей, количество колонок в каждой группе, а также какие модификаторы влияют на выбор колонки, определяется типом клавиши.

Информация, необходимая для построения таблицы символов, содержится в текстовых файлах, образующих «базу данных» XKB. Эта база данных состоит из 5 компонентов:

keycodes — таблицы, которые задают символические имена для скан-кодов клавиш;

types — описание типов клавиш;

compat — описание модификаторов клавиатуры и световых индикаторов на клавиатуре;

symbols — таблицы, в которых для каждого символического имени клавиши (определенного в keycodes) перечисляются все значения, которые должна выдавать клавиша;

geometry — описание расположения клавиш на клавиатуре («геометрия» клавиатуры).

Файлы базы данных XKB находятся в подкаталогах корневой директории XKB, имена которых совпадают с именами компонентов. Каждый файл может содержать несколько поименованных блоков.

Для корректной работы ввода с клавиатуры в конфигурационном файле X-сервера прописывают настройки XKB. Настройки конкретного компонента указывают в виде **имя_файла(имя_блока)**; это означает, что при построении таблицы символов описание данного компонента будет взято из блока с именем **имя_блока** в файле **имя_файла**, который находится в соответствующем подкаталоге корневой директории XKB. Если указано только имя файла, выбирается блок, помеченный флагом **default**. Допускается указывать несколько блоков, разделенных символами «+»; в этом случае описание компонента выбирается из нескольких файлов.

Обычно при конфигурировании X-сервера настройки клавиатуры указываются не перечислением компонентов, а с помощью задания параметров **rules**, **model**, **layout**, **variant** и **options**, которые имеют следующий смысл:

- **model** (модель) определяет тип клавиатуры;
- **layout** (схема) определяет, символы какого алфавита будут соответствовать клавишам;
- **variant** (вариант) задает способ размещения знаков алфавита, заданных параметром **layout**;
- **options** (опции) определяют дополнительные параметры (способ переключения раскладок клавиатуры, значения клавиатурных индикаторов и т.д.);

- **rules** (правила) задает файл, содержащий таблицу правил, в соответствии с которыми по указанным значениям **model**, **layout** и т.д. определяются настройки всех 5 компонентов XKB.

Файлы правил находятся в подкаталоге *rules* корневого каталога XKB. Правила задаются в виде таблиц, причем каждая таблица начинается с шаблона, который определяет, как интерпретируются перечисленные ниже строки. Шаблон начинается с символа «!» Например, шаблон «! model layout = symbols» значениям параметров **model** и **layout** ставит в соответствие значение компонента **symbols**.

Для настройки XKB необходимо указать как минимум параметры **rules**, **model** и **layout**. В параметре **layout** допускается указывать несколько значений через запятую; таким образом, устанавливается несколько раскладок клавиатуры. Вариант иногда указывают в скобках после схемы: **layout(variant)**. Для русского языка необходимо также указывать способ переключения клавиатуры в параметре **options**.

Настройки XKB можно также установить «на лету», используя команду *setxkbmap*. Параметры **rules**, **model**, **layout**, **variant** и **options** задаются с помощью ключей *-rules*, *-model*, *-layout*, *-variant* и *-options* соответственно. Если какой-то параметр не указан, то выбирается его текущее значение. Для отладки полезно указывать ключ *-v level*, включающий вывод дополнительной информации. Параметр level принимает значения от 1 до 10, большее значение соответствует большему количеству выводимой информации. В частности, команда *setxkbmap -v* отображает текущие настройки компонентов XKB.

Для проверки работы XKB полезна также команда *xkbcomp -xkb $DISPLAY test.xkb*. Эта команда запрашивает у X-сервера полные текущие установки XKB (соответствие скан-кодов именам клавиш, описание типов, таблицу символов и др.), преобразует их в текстовый формат и выводит в файл *test.xkb*.

Подробнее о командах *setxkbmap* и *xkbcomp* можно узнать из справочных страниц (*man setxkbmap*, *man xkbcomp*).

2. Пример настройки XKB

Проиллюстрируем все вышеописанное на следующем примере. Пусть настройки XKB прописаны в конфигурационном файле X-сервера */etc/X11/xorg.conf* в разделе **InputDevice** следующим образом:

```
Section "InputDevice"
  ...
  Option       "XkbRules" "xfree86"
  Option       "XkbModel" "pc105"
  Option       "XkbLayout" "us,ru"
  Option       "XkbOptions" "grp:ctrl_shift_toggle"
EndSection
```

Здесь в качестве модели указана 105-клавишная клавиатура, определены английская и русская раскладки клавиатуры; задано переключение ракладок по

нажатию комбинации клавиш Ctrl + Shift. Определение настроек компонентов XKB будет производится с помощью файла правил *xfree86*.

В этом случае команда *setxkbmap -v* выведет на экран следующую информацию:

```
keycodes:   xfree86+aliases(qwerty)
types:      complete
compat:     complete
symbols:    pc(pc105)+us+ru:2+group(ctrl_shift_toggle)
geometry:   pc(pc105)
```

Отсюда видно, каковы настройки компонентов XKB. Особый интерес представляет значение параметра **symbols**. Описание раскладки клавиатуры состоит из нескольких блоков, находящихся в файлах *pc*, *us*, *ru* и *group* в подкаталоге *symbols* корневой директории XKB. Блок **pc(pc105)** описывает поведение служебных клавиш 105-клавишной клавиатуры (функциональных клавиш, клавиш редактирования и т.д.). Блок **us** устанавливает английскую раскладку клавиатуры. Блок **ru** добавляет русскую раскладку клавиатуры как вторую группу символов, на что указывает цифра 2 после двоеточия [2]. Наконец, блок **group(ctrl_shift_toggle)** определяет комбинацию Ctrl+Shift в качестве переключателя раскладок клавиатуры.

Текущие установки XKB можно просмотреть с помощью команды *xkbcomp -xkb $DISPLAY*. Таблица символов при этом находится в секции **xkb_symbols**. Ниже приведено описание одной из алфавитно-цифровых клавиш:

```
key <AD01> {
  type= "ALPHABETIC",
  symbols[Group1]= [               q,               Q ],
  symbols[Group2]= [ Cyrillic_shorti, Cyrillic_SHORTI ]
};
```

Отсюда видно, что с клавишей связано две XKB-группы, причем в первой группе содержатся строчная и заглавная буквы «q», а во второй — строчная и заглавная русские буквы «й». Клавиша относится к типу **ALPHABETIC**, поэтому для выбора колонки используется модификатор Shift [2].

Для переключения раскладок клавиатуры в XKB существуют специальные символы: **ISO_Next_Group** и **ISO_Prev_Group**. Первый из них увеличивает значение текущего номера группы на 1, а при достижении максимального значения устанавливает его в 1. Второй аналогичным образом уменьшает текущий номер группы.

В рассматриваемой таблице символ **ISO_Prev_Group** присутствует в описании клавиш <**LCTL**> (левый Ctrl) и <**LFSH**> (левый Shift), причем в обоих случаях во второй колонке. Для клавиши <**LCTL**> за выбор колонки отвечает модификатор Shift, а клавиша <**LFSH**> относится к типу **PC_BREAK**, в котором для выбора колонки используется модификатор Ctrl. Следовательно, символ **ISO_Prev_Group** будет генерироваться левой клавишей Ctrl при прижатой (левой или правой) клавише Shift либо левой клавишей Shift при

прижатой клавише Ctrl. Аналогично, правые Ctrl и Shift генерируют символ **ISO_Next_Group**. Таким образом, раскладки клавиатуры переключаются комбинацией клавиш Ctrl+Shift, как и прописано в конфигурации XKB.

В современных версиях Linux для настройки XKB используют службу HAL (Hardware Abstract Level, уровень аппаратных абстракций) [3]. Эта служба предоставляет программам-клиентам информацию об имеющихся на компьютере устройствах и их настройках. Конфигурация XKB каким-либо способом заносится в базу данных HAL. Например, в Ubuntu Linux 9.04 в настройках HAL указано, что при обнаружении клавиатуры должен запускаться скрипт *debian-setup-keyboard*. Этот скрипт добавляет в базу данных HAL настройки XKB, которые берет из файла */etc/default/console-setup*.

3. XKB на клиентах SunRay

В процессе администрирования сети ФКН возникла задача настройки XKB на тонких клиентах Sun Ray 2, подключенных к серверу SunFire X4200. Сервер работает под управлением ОС Linux (Ubuntu 9.04). Для работы с тонкими клиентами на нем установлено серверное ПО Sun Ray 4.2.

Следует отметить, что на сервере и на клиентах запускаются разные X-серверы. На сервере работает стандартный Xorg, для которого настройки XKB находятся в каталоге */usr/share/X11/xkb*, а на клиентах запускается Xnewt, специально разработанный Sun для работы на тонких клиентах, для которого корневым каталогом XKB является */opt/SUNWut/lib/xkb*. Поддержка XKB в Xnewt по умолчанию выключена.

В описании установки ПО Sun Ray на сервер под управлением Ubuntu Linux [4] рекомендуется выполнить следующие действия, касающиеся XKB:

- переименовать каталог *xkb* в директории */opt/SUNWut/lib*, в *xkb.bak* и создать вместо него символическую ссылку с именем *xkb* на корневой каталог XKB для Xorg (*/usr/share/X11/xkb*), чтобы Xorg и Xnewt использовали единую базу данных XKB;
- скопировать файл *xkbtable.map* из каталога *xkb.bak* в каталог *xkb*;
- включить поддержку XKB в Xnewt.

После выполнения этих действий возникла проблема: на клиентах вводятся только русские буквы, в то время как на сервере имеются английская и русская раскладки и переключение между ними нормально работает.

В результате анализа этой проблемы выяснилось следующее. При запуске Xnewt запрашивает у ядра Linux аппаратные тип и схему (layout) используемой клавиатуры (см. [5] и комментарии в файле *xkbtable.map*). Затем он производит поиск настроек XKB, соответствующих данным типу и схеме, в файле */opt/SUNWut/lib/xkb/xkbtable.map*. В этом файле для каждой пары «тип, схема» указаны либо готовая раскладка клавиатуры (**keymap**), либо модель и схема XKB. Для определения настроек XKB по найденным модели и схеме используется файл правил *xorg* из каталога */opt/SUNWut/lib/xkb/rules*. В данной версии Ubuntu Linux этот файл является символической ссылкой на файл *base*.

В рассматриваемом случае используется USB-клавиатура Sun Type 6 с русской раскладкой, аппаратный тип которой 6, а схема — 23. По информации из файла *xkbtable.map* можно определить, что этим значениям соответствуют XKB-модель **sun_type6_euro_usb** и схема **ru**. Однако в файле правил *base* модель **sun_type6_euro_usb** не упоминается, поэтому во всех таблицах правил будут выбраны те строчки, в которых в поле **model** указан символ «*» (значения по умолчанию). В результате настройки компонентов XKB будут следующими:

```
keycodes = xfree86+aliases(qwerty)
types = complete
compat = complete
symbols = pc+ru
geometry = pc(pc104)
```

Именно такие значения выдает в данном случае команда *setxkbmap -v*. Из значения параметра **symbols** следует, что на клавиатуре действительно присутствует только русская раскладка, так как блок **pc** определяет поведение служебных клавиш (клавиш редактирования, функциональных клавиш и т.д.). Кроме того, модель клавиатуры не соответствует реально используемой, поэтому не будут работать дополнительные клавиши клавиатуры Sun Type 6 (клавиши управления звуком и др.)

В [6] для решения проблемы с вводом английских букв рекомендуется указать в файле *xkbtable.map* для типа 6 и схемы 23 значение схемы XKB **us** вместо **ru**, однако это приводит к отсутствию русской раскладки на клавиатуре.

На наш взгляд, более корректный способ решения этой проблемы заключается в том, чтобы добавить в файл *base* правила для модели клавиатуры **sun_type6_euro_usb**, аналогичные тем, которые имеются в файле *xorg* каталога */opt/SUNWut/lib/xkb.bak/rules* (непосредственно перенести правила из файла *xorg* в файл *base* нельзя, так как имена файлов и подкаталогов в директориях *xkb* и *xkb.bak*, а также имена блоков в файлах могут не совпадать).

Ниже приводится список внесенных в файл *base* изменений.

1. В таблицу, определенную шаблоном **! model = keycodes**, добавляется правило

   ```
   sun_type6_euro_usb = sun(type6tuv_usb)
   ```

2. В таблицу **! model = geometry** добавляем правило

   ```
   sun_type6_euro_usb = sun(type6tuv)
   ```

3. В таблице с шаблоном **! model layout = symbols** дополнительно указываем следующие правила

   ```
   sun_type6_euro_usb  us = sun_vndr/us(type6)+us
   sun_type6_euro_usb  *  =
                  sun_vndr/us(type6)+us+sun_vndr/%l(type6):2
   ```

Первое из них применяется для схемы **us**. Блок **sun_vndr/us(type6)** определяет поведение служебных клавиш клавиатуры Sun, а **us** устанавливает английскую раскладку клавиатуры. (Блок **sun_vndr/us(type6)** неявно включает английскую раскладку клавиатуры, но проверка показала, что это включение по непонятным причинам не производится). Второе правило срабатывает для схем, отличных от **us**. Первые два блока действуют аналогично предыдущему случаю. В третьем блоке метасимвол **%l** заменяется на значение параметра **layout**, то есть добавляется соответствующая национальная раскладка клавиатуры, как вторая группа символов.

4. В таблицу **! model layout[1] = symbols** нужно добавить правило

```
sun_type6_euro_usb  *  = sun_vndr/us(type6)+ %l[1]%(v[1])
```

Это необходимо для корректного поиска настроек XKB в случае, когда в **layout** указано несколько значений через запятую. Здесь метасимволы **%l[1]** и **%(v[1])** заменяются соответственно на первое значение в параметре **layout** и на первый вариант (если он указан) [2]. Таким образом, это правило определяет поведение служебных клавиш и национальную раскладку клавиатуры, соответствующую первой указанной схеме. Раскладки для остальных значений параметра **layout** добавляются правилами из таблиц **! layout[2] = symbols**, **! layout[3] = symbols**, **! layout[4] = symbols**.

После внесения этих изменений в файл *base* русскую раскладку клавиатуры на клиентах Sun Ray можно включить, указав в файле *xkbtable.map* для типа 6 и схемы 23 в качестве значения схемы XKB **ru** либо **us,ru**. Проверка установок XKB, полученных командой *xkbcomp -xkb $DISPLAY*, показывает, что в обоих случаях русская раскладка клавиатуры добавляется как вторая группа символов. Кроме того, начинают работать клавиши управления звуком.

Вышеперечисленные правила не определяют способ переключения раскладок клавиатуры. Это можно сделать с помощью команды *setxkbmap* с ключом *-options*, например:

```
setxkbmap -option "grp:ctrl_shift_toggle"
```

Эту команду можно вставить в скрипт, срабатывающий при входе пользователя в систему (например, *$HOME/.profile* или *$HOME/.bashrc*).

Литература

1. Pascal, I.U. X Keyboard Extension [Электронный ресурс] / I.U. Pascal. – Режим доступа: http://pascal.tsu.ru/other/xkb
2. Toman, K. How to further enhance XKB configuration [Электронный ресурс] / K. Tomas, I.U. Pascal. – Режим доступа: http://www.xfree86.org/current/XKB-Enhancing.html
3. HAL (freedesktop.org) [Электронный ресурс] – Режим доступа: http://ru.wikipedia.org/wiki/HAL_(freedesktop.org)

4. SRSS 4.1 on Ubuntu Server (i386 and amd64) [Электронный ресурс] – Режим доступа: http://wiki.sun-rays.org/index.php/Sun_Ray_on_Ubuntu
5. Re: Weird keyboard layout problems with SRSS4/Ubuntu 7.10 [Электронный ресурс] – Режим доступа: http://osdir.com/ml/os.solaris.sunray/2008-01/msg00180.html
6. Sun Ray Server [Электронный ресурс] – Режим доступа: http://www.nest.su/tag/sun-ray-server

Математические структуры и моделирование
2009, вып. 20, с. 121–133

УДК 004.925

ВИЗУАЛИЗАЦИЯ СКАЛЯРНОГО ПОЛЯ НА ОСНОВЕ ДИНАМИЧЕСКОГО ПОСТРОЕНИЯ ИЗОПОВЕРХНОСТИ

А.Ю. Суравикин, В.В. Коробицын

Предложена реализация метода построения изоповерхностей скалярного поля на графическом процессоре. Для построения поверхности применен метод граничных тетраэдров. Описан способ кодирования информации о тетраэдрах в текстурах, обрабатываемых на графическом устройстве. Представлены листинги вершинного и геометрического шейдеров. Приведены результаты тестирования производительности реализации алгоритма.

Введение

При решении задач моделирования различных физических процессов часто приходится оперировать трехмерными массивами данных. Например, при моделировании частиц газов или жидкостей можно представить некоторый ограниченный объем как резервуар, в котором для каждой точки определено значение давления газа [1]. Приведенный в примере объем можно рассматривать как скалярное поле, в каждой точке которого определяется количество частиц. В ходе моделирования газ или жидкость будет перемещаться внутри резервуара, и, следовательно, количество частиц в определенной области будет изменяться. Для подобных экспериментов имеет смысл визуализировать количество жидкости в единице объема.

Существует несколько способов визуализации жидкости. Один из них основан на прямой визуализации трехмерного поля и использует технологии, подобные Volume Ray Marching (Casting), т.е. происходит трассировка лучей сквозь резервуар (объем) и при формировании изображения рассчитывается столкновение луча с объектом или попаданием луча в некую окрестность частицы. От плотности может зависеть цвет или уровень прозрачности жидкости в данной точке. Подробнее о Ray Marching можно узнать в [2], [3] и [4]. В данной статье представлен другой способ: он основан на создании поверхности, которая ограничивает жидкость с заранее заданной плотностью. Этот способ отличается

Copyright © 2009 **А.Ю. Суравикин, В.В. Коробицын.**
Омский государственный университет.
E-mail: tizona_del_cid@rambler.ru

тем, что вместо отображения собственно поля с помощью попиксельных алгоритмов используется создание поверхности из примитивов. Следовательно, изменяется область применения: вместо визуализации внутренней структуры жидкости изображается граница раздела сред.

Поскольку алгоритмы визуализации жидкостей и газов достаточно ресурсоёмкие, то целесообразно реализовывать эти алгоритмы на скоростных графических процессорах (GPU). Для реализации алгоритмов на GPU обычно используются шейдерные языки, обеспечивающие функционалом программирования графических процессоров.

Цель работы состоит в реализации алгоритма граничных тетраэдров на графическом процессоре. Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучить литературные источники по теоретическим основам алгоритма.
2. Создать конвертор результатов моделирования жидкости в скалярное поле.
3. Визуализировать скалярное поле с помощью геометрических шейдеров.

Дополнительной задачей является выявление сильных и слабых сторон реализации алгоритма на геометрических шейдерах с точки зрения производительности.

Работа включает в себя как реализацию алгоритма Marching Tetrahedra (граничные тетраэдры), так и создание приложения-основы (так называемого framework) для рендера поверхности и моделирования системы частиц и трехмерного скалярного поля. Подробно описан алгоритм и способ его реализации с использованием вершинной и геометрической стадий, оценена скорость визуализации. Актуальность работы заключается в разработке новой программы с помощью уже известного способа визуализации, который не был до этого достаточно детально описан в литературе.

1. Алгоритм граничных тетраэдров (Marching Tetrahedra Algorithm)

1.1. Изоповерхность

Извлечение изоповерхности — это способ создания полигонального представления непрерывного поля, например таких, которые создаются математическим описанием неявных поверхностей или прямыми измерениями, например медицинским сканированием. Изоповерхность изображает поверхность в объеме, для которого все точки имеют одинаковое значение поля, назовем его изозначением. Для различных изозначений создаваемая поверхность будет также различной (подробно описано в [4] и [5]).

Алгоритм создает изоповерхность, заполняя пространство поля плотно расположенными тетраэдрами. Для каждой вершины тетраэдра значение поля в этой точке сравнивается с изозначением, чтобы получить булево значение. Если

это значение разное для вершин одного тетраэдра, то изоповерхность проходит через объем, ограниченный этим тетраэдром. Если предположить, что поверхность плоская внутри тетраэдра, то сечением тетраэдра этой плоскостью будет либо треугольник, либо четырехугольник.

1.2. Линии с информацией о смежности

Линии со смежностью — новый тип примитивов, который состоит из отрезка и двух дополнительных вершин. Таким образом, в геометрическом шейдере обрабатывается одна такая линия, состоящая из четырех трансформированных вершин.

Отрезок рисуется от вершины $(4i+2)$ к $(4i+3)$ для каждого значения $i = 0, 1, ...n-1$ при общем количестве линий n. Для i-го отрезка вершины $(4i+1)$ и $(4i+4)$ будут смежными соответственно к вершинам $(4i+2)$ и $(4i+3)$.

На рисунке 1 схематично изображен порядок формирования таких примитивов. Подробнее о линиях со смежностью и других новых видах примитивов можно узнать в [9].

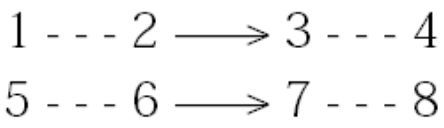

Рис. 1. Линии с дополнительными вершинами. Вершины основных примитивов соединены сплошной линией, прерывистой линией соединены соседние вершины, которые могут использоваться в геометрическом шейдере

1.3. Реализация на графическом процессоре

Извлечение изоповерхности реализуется на графическом процессоре с помощью совместного использования вершинного и геометрического шейдеров (общую информацию о шейдерах можно найти в [10]). Основой будет точная тетраэдризация объема, где четыре вершины каждого тетраэдра передаются как линия с двумя смежными вершинами. Расскажем об этих объектах подробнее.

Вершинный шейдер считывает значения поля в вершинах тетраэдра, сравнивает его с изозначением, а также трансформирует вершину в экранные координаты. Геометрический шейдер обрабатывает результаты работы вершинного шейдера со всех четырех вершин тетраэдра. Он производит операцию логического ИЛИ результатов сравнений с изозначением и формирует четырехбитовое целое значение. Оно определяет геометрию, которую произведет геометрический шейдер. Если все биты установлены в 1 (получим значение 15) или в 0 (получим значение 0), то тетраэдр не пересекает изоповерхность и геометрия не создается. В противном случае, четырехбитное значение используется как координата текстуры, которая содержит таблицу значений, описывающих какую геометрию создаст шейдер.

Таблица 1. Таблица соответствия индекса и номеров ребер тетраэдра

индекс	ребра 0 и 1	ребра 2 и 3
0	0, 0, 0, 0	0, 0, 0, 1
1	3, 0, 3, 1	3, 2, 0, 0
2	2, 1, 2, 0	2, 3, 0, 0
3	2, 0, 3, 0	2, 1, 3, 1
4	1, 2, 1, 3	1, 0, 0, 0
5	1, 0, 1, 2	3, 0, 3, 2
6	1, 0, 2, 0	1, 3, 2, 3
7	3, 0, 1, 0	2, 0, 0, 0
8	0, 2, 0, 1	0, 3, 0, 0
9	0, 1, 3, 1	0, 2, 3, 2
10	0, 1, 0, 3	2, 1, 2, 3
11	3, 1, 2, 1	0, 1, 0, 0
12	0, 2, 1, 2	0, 3, 1, 3
13	1, 2, 3, 2	0, 2, 0, 0
14	0, 3, 2, 3	1, 3, 0, 0

Из таблицы 1 мы получаем восемь значений, которые описывают номера вершин четырех ребер, пересекающих изоповерхность. Значения для четвертого ребра закодированы специальным образом: первая вершина (седьмая колонка значений в таблице) всегда будет ненулевой, если пересечение является четырехугольником. Мы используем эту особенность как условие, генерировать четвертую вершину (а значит, и второй треугольник) или нет. Для создания выходного примитива шейдер решает линейное уравнение на трех или четырех ребрах. Таким образом, происходит интерполяция позиции вершины к точке, где поле пересекает ребро. Это выходные значения для сгенерированных примитивов.

1.4. Настройка по адресам (Swizzling)

В дополнение к реализации прямого извлечения изоповерхности в данном проекте используется техника для оптимизации проходов объема. Вместо того, чтобы производить рендер сетки тетраэдров в построчном порядке, мы изменяем их порядок для улучшения локальности обращений. Техника основана на изменении порядка битов: при проходе объем делится на множество из 8 маленьких объемов (расположение 2 x 2 x 2), каждый из которых содержит шесть тетраэдров. Далее производится проход по этим блокам в обычном построчном порядке. Возможны также другие способы проходов, но используемый метод представляет собой компромисс между сложностью и производительностью.

2. Описание алгоритма

Алгоритм можно разделить на две основные части — инициализацию и рендер, т.е. расчет и вывод изображения. Рассмотрим эти этапы более детально.

2.1. Инициализация

Прежде чем строить изоповерхность, необходимо разбить пространство на ячейки, в узлах которых будет считываться значение поля. Для упрощения расчетов на графическом процессоре ограничим количество узлов числами, являющимися степенями 2. Собственно инициализация состоит из следующих этапов:

1. Создание трехмерной решетки.
2. Загрузка программы для графического процессора.
3. Создание таблицы границ для ячейки.

Программы загружаются с использованием функций графического API, при загрузке передаем параметры, не меняющиеся во время выполнения шейдерных программ. Решетка создается из вершин, расположенных с одинаковыми интервалами по кубу со значениями [-1; 1) по всем измерениям. Индексы создаются для ячеек, в узлах которых расположены созданные вершины. Каждая ячейка состоит из 6 тетраэдров, в каждом тетраэдре 4 индекса. Индекс формируется побитово из целочисленных значений положения ячейки в решетке:

```
index=(int)((x)|((y)<<sizeLog2[0])|((z)<<(sizeLog2[0]+sizeLog2[1])))
```

Здесь x, y, z - положение ячейки, $sizeLog2$ - логарифм по основанию 2 числа ячеек в стороне решетки.

2.2. Рендер

Реализация рендера жидкости описана далее поэтапно в соответствии с порядком обработки данных на графическом конвейере. Описаны вершинный и геометрический шейдеры, этап растеризации и пиксельный шейдер тривиальны.

2.2.1. Вершинный шейдер

Графический процессор рендерит вершинный буфер, загруженный на этапе инициализации и обрабатывает вершины этого буфера-сетки. Вершинная программа приведена в листинге 1. В коде приведены комментарии, объясняющие последовательность действий.

Листинг 1. Программа обработки вершин

```
// Vertex shader
// march_tetra.vert
uniform mat4 WorldViewProj; // Матрица трансформации
uniform sampler3D FieldTex; // Сэмплер текстуры-источника данных

// Маска и смещение битов для сетки
uniform ivec3 SizeMask;
uniform ivec3 SizeShift;
```

```
uniform vec3 scalePos; // Фактор масштабирования
uniform float IsoValue; // Изозначение

void main() //Главная функция
{
    int index = gl_VertexID;//Получаем индекс текущей вершины

    // Получаем позицию считывания данных текстуры из индекса
    vec3 Pos;
    Pos.x = float((index >> SizeShift.x)&SizeMask.x)/(SizeMask.x+1);
    Pos.y = float((index >> SizeShift.y)&SizeMask.y)/(SizeMask.y+1);
    Pos.z = float((index >> SizeShift.z)&SizeMask.z)/(SizeMask.z+1);
    // Считываем значение тесктуры в рассчитанной позиции
    vec4 Field = tex3D(FieldTex, Pos);
    Pos = Pos * 2.0 - 1.0;
    // Масштабируем позицию
    Pos = Pos * scalePos;

    // Трансформируем позицию
    gl_Position = mul(WorldViewProj, vec4(Pos, 1.0));
    // Передаем на следующую стадию значение поля и
    // булево значение сравнения
    gl_TexCoord[0].x =  Field.x;
    gl_TexCoord[0].y = (Field.x > IsoValue) ? 1.0 : 0.0;
}
```

При обработке массива вершин мы используем следующие входные данные:

1. Индекс вершины, т.е. ее номер ($index$). По этому номеру рассчитывается положение вершины в сетке (ячейках скалярного поля) и приводится к интервалу $[0; 1)$ для считывания из текстуры.

2. 3D-текстура $FieldTex$, содержащая значения скалярного поля.

3. Вспомогательные константы для пространственных преобразований: $SizeMask$, $SizeShift$, $scalePos$, а также матрица $WorldViewProj$ (для преобразования позиции вершины в экранные координаты).

4. Изозначение ($IsoValue$), по которому будет строиться поверхность.

Вершинный шейдер возвращает трансформированную вершину ($gl\_Position$), а также значение поля в этой точке ($gl\_TexCoord[0].x$) и булево значение сравнения с IsoValue ($gl\_TexCoord[0].y$). При этом для изменения размеров сетки «резервуара» используется покомпонентное умножение вектора позиции на вектор $scalePos$ (в нашем примере $\{30.0; 30.0; 30.0\}$).

В случае размеров сетки 64x64x64 (в данной реализации размер сетки должен быть степенью 2) значение $SizeMask$ устанавливается в $\{63; 63; 63\}$ ($2^n - 1$,

где степень $n = 6$). $SizeShift$ равен n, в данном случае 6. Переменную $IsoValue$ (изозначение) установим в 0.15 (при этом «вес» каждой частицы равен 0.1). $WorldViewProj$ — матрица преобразования позиции вершины в экранные координаты.

2.2.2. Геометрический шейдер

Обработанные вершины группируются в примитивы, состоящие из четырех вершин (см. выше). Вершины по расположению образуют тетраэдр. Программа обработки представлена в листинге 2.

Листинг 2. Программа обработки примитивов

```
// Geometry shader
// march_tetra.geom
// входные данные: Линии с соседями (тетраэдры)
// выходные данные: 0, 1 или 2 треугольника в зависимости
//                  от сечения изоповерхностью тетраэдра

uniform sampler2DRect edgeTex; //текстура-таблица ребер
uniform float IsoValue; //изозначение

// Функция расчета точек пересечения изоповерхности и заданного ребра
// Используется линейная интерполяция (функция lerp()) между вершинами
// ребра по изозначению и значениям поля в точках, соответствующих
// этим вершинам
vec4 CalcIntersection(vec4 Pos0, vec2 Field0, vec4 Pos1, vec2 Field1)
{
  float t = (IsoValue - Field0.x) / (Field1.x - Field0.x);
  return lerp(Pos0, Pos1, t);
}

// Расчет нормали к плоскости, заданной тремя точками
// Используется векторное произведение двух векторов,
// соединяющих заданные вершины
vec3 calcNormal(vec3 v0, vec3 v1, vec3 v2)
{
  vec3 edge0 = v1 - v0;
  vec3 edge1 = v2 - v0;
  return normalize(cross(edge0, edge1));
}

void main()
{
  // Из полученных булевых значений собираем индекс тетраэдра
  int index = (int(gl_TexCoordIn[0][0].y) << 3) |
              (int(gl_TexCoordIn[1][0].y) << 2) |
```

```
            (int(gl_TexCoordIn[2][0].y) << 1) |
             int(gl_TexCoordIn[3][0].y);

// Если тетраэдр полностью лежит снаружи (index = 0) или внутри
// (index = 15) поверхности, то для него не создаем треугольники
if (index > 0 && index < 15)
{
  //Получаем значения из таблицы ребер для текущего значения index
  ivec4 e0 = ivec4(texelFetch2DRect(edgeTex, ivec2(index,0))*255);
  ivec4 e1 = ivec4(texelFetch2DRect(edgeTex, ivec2(index,1))*255);

  // Создаем треугольник из точек пересечения изоповерхности
  // и текущего тетраэдра
  vec4 p0 = CalcIntersection(
           gl_PositionIn[e0.x], gl_TexCoordIn[e0.x][0].xy,
           gl_PositionIn[e0.y], gl_TexCoordIn[e0.y][0].xy);
  vec4 p1 = CalcIntersection(
           gl_PositionIn[e0.z], gl_TexCoordIn[e0.z][0].xy,
           gl_PositionIn[e0.w], gl_TexCoordIn[e0.w][0].xy);
  vec4 p2 = CalcIntersection(
           gl_PositionIn[e1.x], gl_TexCoordIn[e1.x][0].xy,
           gl_PositionIn[e1.y], gl_TexCoordIn[e1.y][0].xy);

  // Расчет нормали для плоскости созданного треугольника
  vec3 n = calcNormal(p2.xyz, p1.xyz, p0.xyz);
  // Задаем параметры первой вершины и создаем ее
  gl_Position = p0;
  gl_TexCoord[0] = vec4(n, 0.0);
  EmitVertex();
  // Задаем параметры второй вершины и создаем ее
  gl_Position = p1;
  gl_TexCoord[0] = vec4(n, 0.0);
  EmitVertex();
  // Задаем параметры третьей вершины и создаем ее
  gl_Position = p2;
  gl_TexCoord[0] = vec4(n, 0.0);
  EmitVertex();
  // Аналогично создаем второй треугольник,
  // если сечение - четырехугольник
  if (e1.z != 0)
  {
       vec4 p3 = CalcIntersection(
                 gl_PositionIn[e1.z], gl_TexCoordIn[e1.z][0].xy,
                 gl_PositionIn[e1.w], gl_TexCoordIn[e1.w][0].xy);
       n  = calcNormal(p1.xyz, p2.xyz, p3.xyz);
```

```
        gl_Position = p3;
        gl_TexCoord[0] = vec4(n, 0.0);
        EmitVertex();
    }
  }
}
```

В коде геометрического шейдера показано, как происходит создание примитивов, представляющих собой сечения поверхности в каждом тетраэдре. Пример такого сечения показан на рис. 2. Для наглядности пример представлен в двумерном виде.

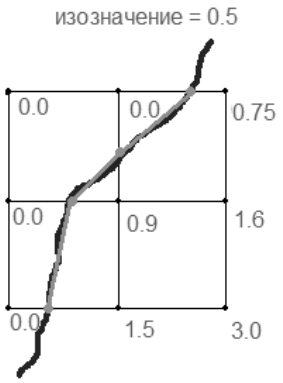

Рис. 2. Сечение поверхностью тетраэдров

Расскажем подробнее о работе геометрического шейдера. Для каждой группы вершин выполняется функция $main()$. При этом размер группы определяется типом входящего примитива. В нашем случае берется 4 вершины, т.к. используемый тип — линии со смежностью. Сначала с помощью побитовых операций формируется индекс ($index$) из булевых значений вершин, затем происходит сравнение индекса с граничными значениями (0 и 15): если индекс равен одному из них, то пропускаем обработку этого примитива, т.к. значение 0 означает, что тетраэдр полностью лежит снаружи объема (ограниченного изоповерхностью), а 15 (1111_2) - что тетраэдр лежит внутри него. Иначе создаем примитивы следующим образом:

1. Получаем массивы номеров ребер $e0$ и $e1$ из таблицы ребер (таблица хранится в отдельной текстуре, хотя можно ее задать через Constant Buffer).

2. Рассчитываем позиции вершин для создаваемого треугольника с помощью линейной интерполяции между вершинами тетраэдров по изозначению поля. Для каждой вершины используется функция $CalcIntersection$, в которой по значению поля в вершинах ребра рассчитывается коэффициент интерполяции, а затем производится собственно интерполяция координат вершин по этому коэффициенту.

3. Расчет нормали к полученному треугольнику. Нормали к плоскости создаваемого треугольника используются для расчета его освещения. В приведенном листинге расчет производится на этапе создания треугольника по трем вершинам (функция *calcNormal*). Однако есть другой способ: использовать скалярное поле, а точнее, градиенты скалярного поля в точках, соответствующих полученным вершинам треугольника. Расчет нормалей по градиентам производится с помощью функции, описанной в Листинге 3.

Листинг 3. Функция расчета нормали как градиента поля

```
// Расчет нормали как градиента поля в точке:
// s - семплер (sampler) текстуры поля,
// t - текстурная координата,
// invFieldTexSize - вектор обратных значений размеров 3D-текстуры поля
vec3 calcGradientNormal(sampler3D s, vec3 t)
{
vec4 step = vec4(invFieldTexSize, 0.0);
vec3 gradient = vec3(
texture3D(s, t + step.xww).x - texture3D(s, t - step.xww).x,
texture3D(s, t + step.wyw).x - texture3D(s, t - step.wyw).x,
texture3D(s, t + step.wwz).x - texture3D(s, t - step.wwz).x);
return normalize(-gradient);
}
```

Используется формула расчета градиента, закодированная таким образом, чтобы достигать высокой производительности на системах с параллельной архитектурой. Недостатком является 6 чтений из текстуры (т.к. это операции с большой латентностью), однако они производятся из соседних ячеек, поэтому будет эффективно использоваться текстурный кэш, который уменьшит задержки на доступ к текстуре.

Вызов данной функции осуществляется для каждой вершины следующим образом:

```
n = calcGradientNormal(FieldTex, lerp(gl_TexCoordIn[e0.x][1].xyz,
   gl_TexCoordIn[e0.y][1].xyz, t));
```

здесь $FieldTex$ - семплер текстуры поля, т.е. переменная, по которой определяется какая текстура и каким образом будет считываться. Вторым параметром идет результат функции *lerp*, т.е. линейной интерполяции значений текстурных координат поля. Остановимся на этом параметре. Это координаты точки в поле, приведенные к интервалу $[0; 1)$, используемые для адресации текстуры. С помощью временной переменной *step* мы можем обращаться к соседним ячейкам текстуры, таким образом рассчитывая градиент поля. Параметры функции интерполяции — значения текстурных координат соответствующих вершин ($gl\_TexCoordIn$[индекс ребра][1] — текстурная координата поля передается из вершинного шейдера) и коэффициент интерполяции t, рассчитанный в функции $CalcIntersection$. Коэффициент интерполяции для позиции вершины и ее

текстурных координат будет одинаковым, поэтому имеет смысл рассчитать его один раз и использовать как при расчете позиции, так и при расчете нормали.

В переменные $gl\_Position$ и $gl\_TexCoord[0]$ заносятся соответственно позиция и нормаль (передается через текстурную координату) для создаваемой вершины, затем происходит вызов функции $EmitVertex$, который формирует вершину. После создания трех вершин, которые сформируют новый треугольник, идет проверка индекса $e1.z$, т.е. первой вершины четвертого ребра тетраэдра. Если значение не равно нулю, то происходит создание четвертой вершины, которая вместе с двумя предыдущими сформирует еще один треугольник. В результате получим один или два треугольника, которые будут аппроксимировать пересечение изоповерхности текущего тетраэдра. Подобная операция производится для всех тетраэдров, таким образом, получим всю изоповерхность.

3. Результаты моделирования

Изображение изоповерхности, построенное в предложенной реализации, приведено на рис. 3. Однако при выводе на экран не производится сглаживание построенной поверхности, поэтому изображение выглядит угловатым. Процедура сглаживания изображения поверхности будет разработана позже.

Рис. 3. Рендер изоповерхности

Необходимо отметить, что этот рисунок является одним кадром из генерируемой анимации процесса движения струи жидкости. Вывод анимации на экран производится в реальном времени, и ее скорость зависит от параметризации модели и производительности графической системы. Для проверки производительности алгоритма мы протестировали его на различных по размеру

массивах данных. В таблице 2 представлены данные производительности, измеренные в кадрах в секунду (Frames Per Second, FPS) для массивов с 32x32x32 по 128x128x128 на двух системах. Так как в реализации понятия размера массива и размера сетки (из которой формируется поверхность) разделены, то можно отдельно регулировать качество визуализации (число треугольников, аппроксимирующих поверхность, напрямую влияет на детализацию) и точность расчетов (чем больше размер текстуры данных, тем выше точность). Это актуально, если дополнительно использовать скалярное поле для собственно моделирования жидкости (расчетов плотности, давления и т.п.). В данной реализации используется упрощенная система моделирования жидкости, основанная на системе частиц, поэтому имеет смысл устанавливать только одинаковые размеры сетки и текстуры.

Производительность реализации алгоритма оценивалась на двух вычислительных системах: система 1 — AMD Phenom X4 9850+, 4GB RAM, NVIDIA GeForce 260 GTX; система 2 — Intel Core 2 Duo 2.26, 4GB RAM, NVIDIA GeForce 9800M GTS.

Таблица 2. Таблица производительности реализации алгоритма на различных размерах массивов данных

Размер поля	Производительность (FPS)	
	система 1	система 2
32x32x32	157,0	112,5
64x64x64	81,5	64,0
128x128x128	17,1	15,5

Из таблицы 2 также видно, что, несмотря на серьезное различие в теоретической производительности видеоподсистем, различия в реальной производительности не настолько значительны и уменьшаются при увеличении размера текстуры. Это сигнализирует о том, что при увеличении нагрузки мы сталкиваемся с т.н. «узким местом» (bottleneck) системы. Способ избавиться от этого еще предстоит выяснить.

Заключение

В работе реализован алгоритм, позволяющий интерактивно отображать результаты моделирования жидкостей, а точнее, границы раздела сред. Визуализация скалярного поля основана на алгоритме граничных тетраэдров. Данный алгоритм был реализован с использованием библиотеки OpenGL. При этом на визуализацию поля разрешением 128x128x128 видеокарте NVIDIA GeForce 9800M GTS требуется около 50–70 мс, что позволяет использовать этот метод в интерактивных приложениях, т.е. при изменении данных скалярного поля можно строить полигональную поверхность несколько раз в секунду.

Возможное улучшение алгоритма состоит в том, чтобы сделать сетку тетраэдров нерегулярной, например, использовать такие структуры как oct-tree, k-d tree и другие. Это позволит увеличивать плотность ячеек с тетраэдрами

в тех местах, где больше плотность частиц (или градиент плотности частиц) поля, и уменьшать там, где частиц нет. Следовательно, работа алгоритма станет более эффективной, т.е. будет улучшено качество получаемой поверхности и уменьшено число требуемых для этого ячеек.

Литература

1. Müller, M. Particle-based fluid simulation for interactive applications / M. Müller, D. Charypar, M. Gross // SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation. – 2003. – P. 154–159.
2. GPU gems 3 / eds. H. Nguyen. – Boston: Pearson Education, 2008. – Доступно также: http://developer.nvidia.com/object/gpu-gems-3.html (5.10.2009).
3. Pawasauskas, J. Volume Visualization With Ray Casting [Электронный ресурс] / J. Pawasauskas // CS563 - Advanced Topics in Computer Graphics (1997). – Режим доступа: http://web.cs.wpi.edu/ matt/courses/cs563/talks/powwie/p1/ray-cast.htm (5.10.2009).
4. NVIDIA OpenGL SDK Guide [Электронный ресурс]. – Режим доступа: http://developer.download.nvidia.com/SDK/10.5/opengl/OpenGL_SDK_Guide.pdf (5.10.2009).
5. Marching Tetrahedra [Электронный ресурс]. – Data Analysis and Assessment Center. – Режим доступа: https://visualization.hpc.mil/wiki/Marching_Tetrahedra (5.10.2009).
6. Bourke, P. Polygonising a Scalar Field [Электронный ресурс] / P. Bourke. – Режим доступа: http://local.wasp.uwa.edu.au/ pbourke/geometry/polygonise/ (5.10.2009).
7. Lorensen, W. Marching cubes: a high resolution 3D surface construction algorithm / W. Lorensen, H. Cline // Computer Graphics. – 1987. – V. 21, N. 4. – P. 163-169.
8. Newman, T.S. A survey of the marching cubes algorithm / T.S. Newman, H. Yi // Computers & Graphics. – 2006. – V. 30, N. 5. – P. 854-879.
9. Brown, P. ARB_geometry_shader4 [Электронный ресурс] / P. Brown, B. Lichtenbelt // OpenGL - The Industry Standard for High Perfomance Graphics. – Режим доступа: http://www.opengl.org/registry/specs/ARB/geometry_shader4.txt (5.10.2009).
10. Определение понятия «Шейдер» [Электронный ресурс]. – Википедия. – Режим доступа: http://ru.wikipedia.org/wiki/Шейдер (5.10.2009).

Математические структуры и моделирование
2009, вып. 20, с. 134–140

УДК 003.26

ИССЛЕДОВАНИЕ ГЕНЕРАТОРОВ ПСЕВДОСЛУЧАЙНЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ, ПОСТРОЕННЫХ НА ОСНОВЕ ХЭШ-ФУНКЦИЙ

М.И. Атмашкин, С.В. Белим

В работе проведен статистический анализ псевдослучайных последовательностей, формируемых на основе алгоритмов хэширования SHA-1 и MD5. В качестве метода исследования выбраны графические тесты.

1. Введение

Генераторы псевдослучайных последовательностей (ПСП) являются неотъемлемыми элементами большого количества вычислительных систем. Генератор псевдослучайных последовательностей (ГПСП) — это алгоритм, генерирующий последовательность чисел, элементы которой почти независимы друг от друга и подчиняются заданному распределению. ГПСП, как и поточные шифры, состоят из внутреннего состояния (обычно размером от 16 бит до нескольких мегабайт), функции инициализации внутреннего состояния ключом или зерном (англ. *seed*), функции обновления внутреннего состояния и функции вывода. Если $\{\gamma_i\}$ — псевдослучайная последовательность, полученная при использовании «хорошего» ГПСП, то три следующие задачи должны быть вычислительно сложными:

1) определение $(i-1)$-го элемента γ_{i-1} последовательности на основе известного фрагмента γ_i, γ_{i+1}, γ_{i+2}, ..., γ_{i+b-1} конечной длины b;
2) определение $(i+1)$-го элемента γ_{i+1} последовательности на основе известного фрагмента γ_{i-b+1}, ..., γ_{i-2}, γ_{i-1}, γ_i конечной длины b;
3) определение ключевой информации по известному фрагменту последовательности конечной длины.

Хеширование (англ. *hashing*) — преобразование входного массива данных произвольной длины в выходную битовую строку фиксированной длины. Такие преобразования также называются хеш-функциями, а их результаты называют хешем, хеш-кодом или дайджестом сообщения (англ. *message digest*). В общем случае однозначного соответствия между исходными данными и хеш-кодом нет.

Copyright © 2009 **М.И. Атмашкин, С.В. Белим.**
Омский государственный университет.
E-mail: belim@univer.omsk.su

Поэтому существует множество массивов данных, дающих одинаковые хеш-коды – так называемые коллизии. Вероятность возникновения коллизий играет немаловажную роль в оценке «качества» хеш-функций.

2. Построение генератора псевдослучайных последовательностей

Формальное описание исследуемого далее ГПСП может быть представлено следующим образом:

1. Внутреннее состояние и его функция обновления

Внутренним состоянием ГПСП, основанного на хеш-функции MD5, является 128-битное хеш-значение H_i и ключевая фраза K произвольной длины (возможно и нулевой). Хеш-значение H_i вычисляется на основе предыдущего хеш-значения H_{i-1} в конкатенации с ключевой фразой:

$$Hi = MD5(H_{i-1}||K).$$

Ключевая фраза задается вначале один раз, но возможна маловероятная ситуация, когда H_i совпадет с H_{i-1}. В этом случае, чтобы избежать «застревания» генератора, следует ввести новую ключевую фразу. Для SHA-1 ситуация выглядит аналогично, за исключением того, что хеш-значение имеет размер 160 бит.

2. Функция инициализации внутреннего состояния зерном

В идеале зерно должно с равной вероятностью принять любое из значений от 0 до $2^{m\check{}1}$, где m – битовая разрядность используемой хеш-функции (на практике достаточно, чтобы каждое значение с ненулевой вероятностью). В данной работе для построения ГПСП использовалась комбинация двух методов: «счетчик тактов процессора» и «взаимодействие между потоками». Время (в наносекундах), прошедшее с момента загрузки системы, может быть получено как значение счетчика тактов процессора в момент инициализации внутреннего состояния, деленное на тактовую частоту процессора и умноженное на 10^9. Младшие 8 разрядов этого времени дадут один достаточно равномерно распределенный байт. Если сгенерировать несколько таких независимых байт (16 — для MD5, 20 — для SHA-1), а затем соединить в одно число, то получим зерно, отвечающее заданным требованиям. Равномерность распределения полученных величин следует из равномерного распределения байтов и способа получения зерна, а независимость следует из Теоремы 6 [2, стр. 65]. Независимости программно можно добиться, например, с помощью взаимодействия трех несимметричных по затратам процессорного времени потоков операционной системы. Первый поток – это основной поток программы (Main thread), при генерации зерна запускающий два других потока («Hard» и «Light»), один из которых состоит из бесконечного цикла, а другой генерирует нужное число байт (MD5 — 16 байт; SHA-1 — 20 байт) на основе 8-ми младших разрядов времени в наносекундах. Независимость этих байт обеспечивается сложным механизмом переключения между потоками, зависящим от большого числа факторов, быстрым изменением счетчика времени, дополнительной передачей управления на каждом шаге

цикла генерации, а также некоторыми примитивами синхронизации, которые не дают генерирующему потоку надолго захватывать процессор.

3. Функция вывода

На выходе генератора будет один бит, находящийся на произвольной (заданной заранее) позиции, отсчитываемой справа в текущем хеш-значении. Например, для MD5 — от 0 до 127, а для SHA-1 — от 0 до 159. «Независимость» этих битов обеспечивается сложностью хеш-функций. Далее эти биты последовательно записываются в файл и анализируются. Если предположить, что они действительно независимы и равномерно распределены, то независимыми и равномерно распределенными (по той же Теореме 6 [2, стр. 65]) будут и блоки из этих битов (в программе тестировались блоки до 16 бит).

3. Тестирование ПСП на случайность

Для сравнения ПСП с истинно случайной последовательностью применяются различные статистические тесты. Случайность – вероятностное свойство, это означает, что свойства случайной последовательности могут быть охарактеризованы и описаны в терминах вероятности. Вероятный результат статистических тестов, применяемых к истинно случайной последовательности, известен априорно и может быть описан в вероятностных терминах. Существует бесконечное число возможных статистических тестов, оценивающих присутствие или отсутствие «образца», который при обнаружении указал бы, что последовательность неслучайна. Поскольку существует много тестов, оценивающих, является ли последовательность случайной или нет, никакой определенный конечный набор тестов не считается на сегодняшний день «законченным».

Статистический тест формулируется для проверки определенной нулевой гипотезы H0 о том, что проверяемая последовательность является случайной. С этой нулевой гипотезой связана альтернативная гипотеза о том, что последовательность неслучайна. Для каждого применяемого теста получают заключение о принятии или отклонении нулевой гипотезы, основываясь на сформированной исследуемым генератором последовательности.

Для каждого теста должна быть выбрана подходящая статистика случайности для принятия или отклонения нулевой гипотезы. Согласно предположению о случайности, такая статистика имеет распределение возможных значений. Теоретическое эталонное распределение этой статистики для нулевой гипотезы определяется математическими методами. Из этого эталонного распределения определяется критическое значение. Во время проведения теста вычисляется значение тестовой статистики. Это значение сравнивается с критическим значением. Если значение тестовой статистики превышает критическое значение, то нулевая гипотеза для случайности отклоняется. В противном случае нулевая гипотеза принимается.

В данной работе был проведен ряд графических тестов, описанных в [3]. Рассмотрим результаты этих графических тестов:

1. Гистограмма распределения элементов

В исследуемой последовательности подсчитывается частота встречаемости

каждого элемента, после чего строится график зависимости числа появлений элементов от их численного представления (ASCII-значение для байтов). Для того чтобы последовательность удовлетворяла свойствам случайности, необходимо, чтобы в ней присутствовали все возможные элементы рассматриваемой разрядности, при этом разброс частот появления символов стремился к нулю. В противном случае последовательность не является случайной. Результаты теста приведены на рисунке 1 и удовлетворяют свойствам случайности.

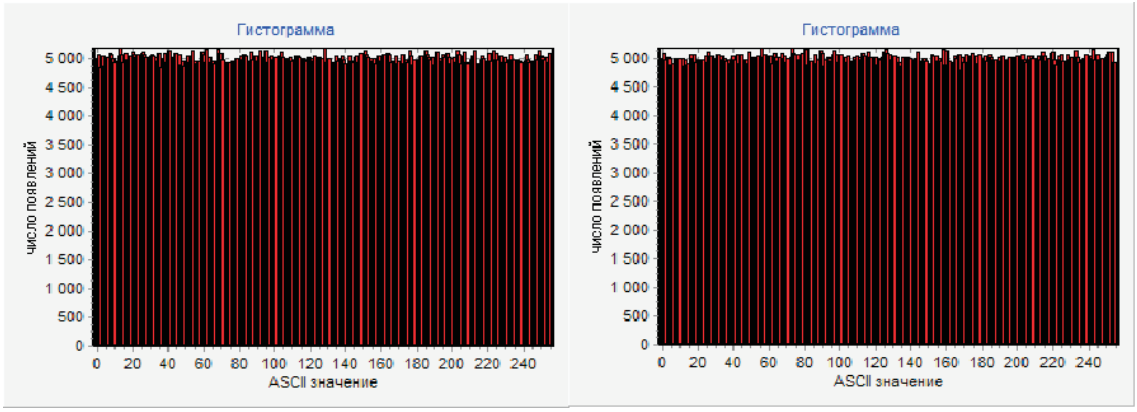

Рис. 1. Гистограмма распределения элементов. MD5 (слева) и SHA1 (справа)

2. Распределение на плоскости

На поле размером $(2^R-1)\times(2^R-1)$ (R – разрядность чисел исследуемой последовательности) наносятся точки с координатами $(\varepsilon_i;\varepsilon_i+1)$, где ε_i – элементы исследуемой последовательности ε, $i=1,\ldots,(n-1)$, n – длина последовательности. Если между элементами последовательности отсутствуют зависимости, то точки на поле расположены хаотично. Если на поле присутствуют зависимости, наблюдаются «узоры» — последовательность не является случайной. Результаты теста приведены на рисунке 2 и удовлетворяют свойствам случайности.

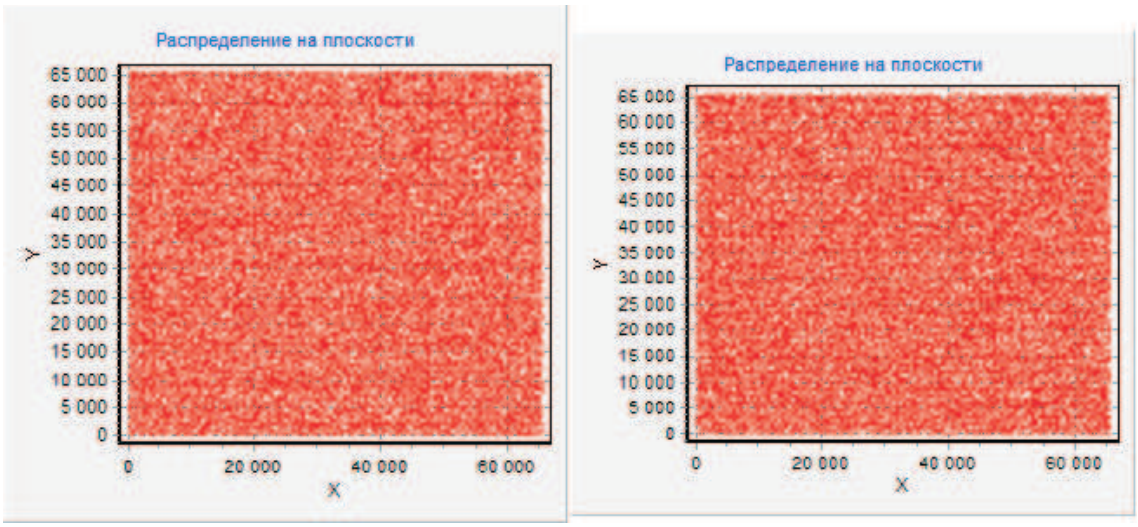

Рис. 2. Распределение на плоскости (R = 16 бит). MD5 (слева) и SHA1 (справа)

3. Проверка серий

Подсчитывается, сколько раз встречаются нули, единицы, серии-двойки (00, 01, 10, 11), серии-тройки (000, 001, 010, 011, 100, 101, 110, 111) и т. д. в битовом представлении исследуемой последовательности. Полученные результаты представляются в графическом виде. У последовательности, чьи статистические свойства близки к свойствам истинно случайной последовательности, разбросы между числом появлений серий каждого вида должны стремиться к нулю. В противном случае последовательность не является случайной. Результаты теста показаны на рисунке 3 и удовлетворяют свойствам случайности.

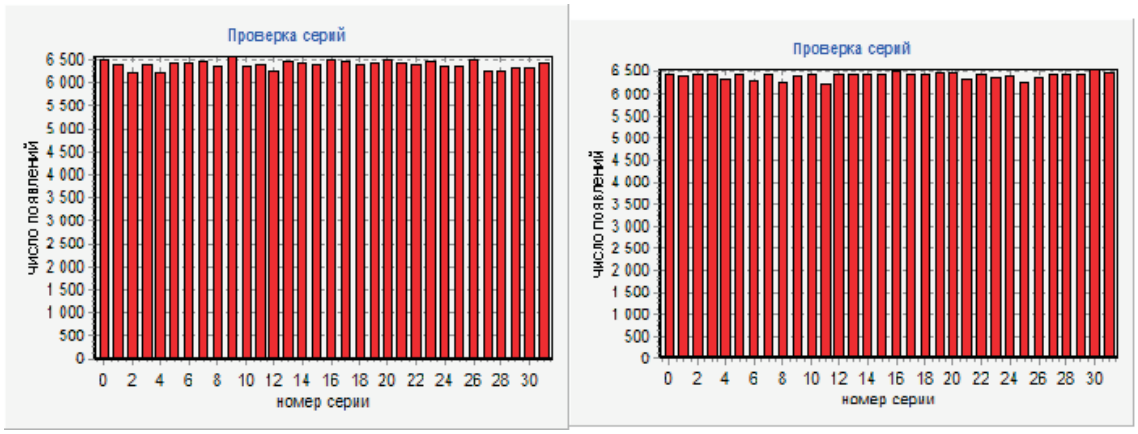

Рис. 3. Проверка серий длиной 5 бит. MD5 (слева) и SHA1 (справа)

4. Проверка на монотонность

Исследуемая последовательность графически представляется в виде следующих друг за другом непересекающихся участков невозрастания и неубывания элементов последовательности. У последовательности, чьи статистические

свойства близки к свойствам истинно случайной последовательности, вероятность появления участка невозрастания (неубывания) определенного размера зависит от его длины: чем больше длина, тем меньше вероятность. В противном случае последовательность не является случайной. Результаты теста показаны на рисунке 4 и удовлетворяют свойствам случайности.

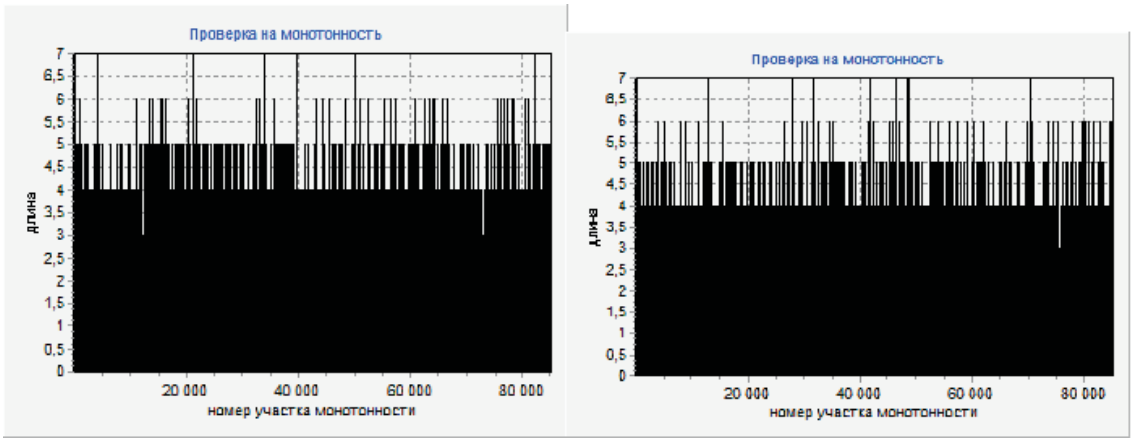

Рис. 4. Проверка на монотонность. MD5 (слева) и SHA1 (справа)

5. Профиль линейной сложности

Пусть $\varepsilon(n) = \varepsilon_1\varepsilon_2...\varepsilon_n$ — двоичная последовательность длины n. Последовательно рассматриваются подпоследовательности $\varepsilon(k)$, содержащие первые k элементов последовательности, и строится график зависимости линейной сложности L от длины подпоследовательности N. Линейная сложность вычисляется по алгоритму Берлекэмпа-Масси, который подробно описан в [1]. У последовательности, чьи свойства близки к свойствам истинно случайной последовательности, линия графика должна стремиться к линии $L = N/2$. В противном случае последовательность не является случайной. Результаты теста настоящего ГПСП показаны на рисунке 5 и удовлетворяют свойствам случайности.

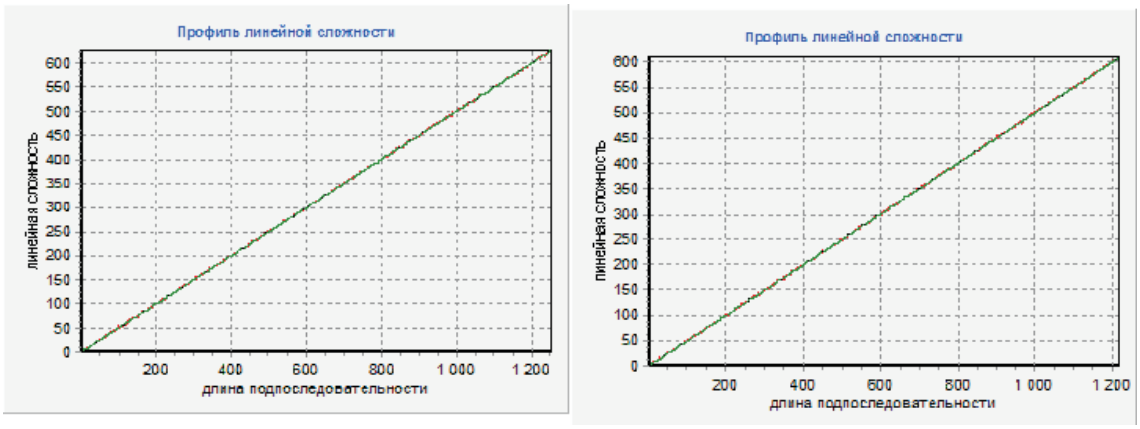

Рис. 5. Профиль линейной сложности. MD5 (слева) и SHA1 (справа)

6. Графический спектральный тест

Пусть $\varepsilon(n) = \varepsilon_1\varepsilon_2...\varepsilon_n$ — двоичная последовательность длины n. Преобразуем ее в последовательность $x = x_1x_2...x_n$, где $x_i = 2 \cdot \varepsilon_i \dot{-} 1$. Затем применим к x дискретное преобразование Фурье и получим последовательность гармоник:

$$S_j = \sum_{k=1}^{n} x_k e^{-i\frac{2\pi j}{n}(k-1)}.$$

Графически изобразим модули этих гармоник. У последовательности, чьи свойства близки к свойствам истинно случайной последовательности, число гармоник, длины которых значительно превышают среднюю длину гармоники, должно стремиться к 0. В противном случае последовательность не является случайной. Результаты теста ГПСП показаны на рисунке 6 и удовлетворяют свойствам случайности (в 3 раза среднюю длину превышают MD5 – 0.04%, SHA1 – 0.08%).

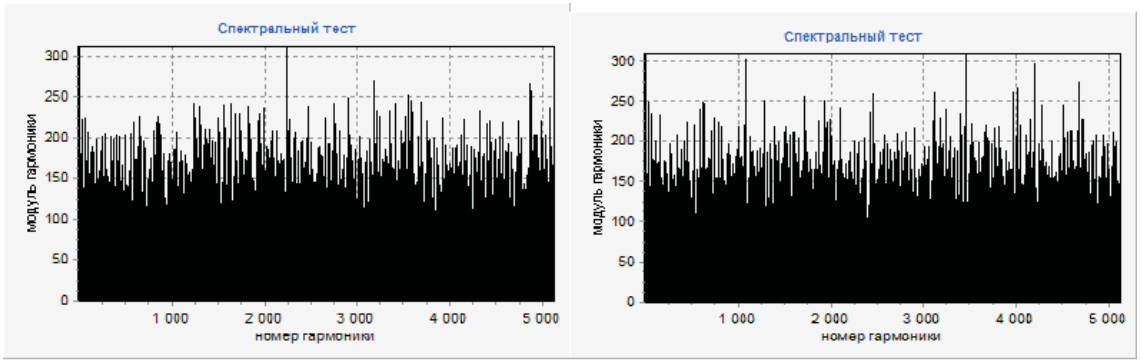

Рис. 6. Графический спектральный тест. MD5 (слева) и SHA1 (справа)

4. Заключение

Таким образом, генератор псевдослучайных чисел, построенный с использованием широко известных алгоритмов хэширования SHA-1 и MD5, является достаточно «хорошим» со статистической точки зрения. Данный факт весьма существенен при построении ряда систем защиты информации. Описанные хэш-функции достаточно легко поддаются аппаратной реализации, что позволяет строить качественные ГПСП на базе чипов с низкой тактовой частотой.

Литература

1. Берлекэмп, Э. Алгебраическая теория кодирования / Э. Берлекэмп. – М.: Мир, 1971. – 479 с.
2. Боровков, А.А. Теория вероятностей / А.А. Боровков. – М.: Наука, 1986. – 432 с.
3. Иванов, М.А. Теория, применение и оценка качества генераторов псевдослучайных последовательностей / М.А. Иванов, И.В. Чугунков. – М.: КУДИЦ-ОБРАЗ, 2003. – 240 с.

Математические структуры и моделирование
2009, вып. 20, с. 141–152

УДК 65.012.8

СОВМЕСТНАЯ РЕАЛИЗАЦИЯ МАНДАТНОГО И РОЛЕВОГО РАЗГРАНИЧЕНИЯ ДОСТУПА К ИНФОРМАЦИИ В КОМПЬЮТЕРНЫХ СИСТЕМАХ

С.В. Белим, Н.Ф. Богаченко, Ю.С. Ракицкий

В статье анализируется возможность совмещения ролевой и мандатной политик безопасности на основе графовой модели. С этой целью вводится понятие решеточного дерева и обобщается модель ролевого разделения доступа. Приводится простейший алгоритм мандатного разделения доступа, учитывающий концепцию ролей.

1. Введение

Необходимость совмещения различных типов разграничения доступа к информации в корпоративных сетях, как правило, обусловлена требованиями политики безопасности предприятия к хранению и обработке данных. На сегодняшний день общепринятой практикой стало использование систем управления базами данных для организации доступа к ресурсам. Все современные базы данных используют концепцию ролей для выдачи полномочий пользователям, реализуя таким образом ролевое разграничение доступа. Однако в ряде организаций, особенно связанных с защищенным документооборотом, также налагается требование использования меток безопасности и, основанного на них, мандатного разделения доступа. В рамках мандатного разделения доступа множество разрешенных доступов задается неявным образом в виде уровня конфиденциальности для объектов и уровня доверия для субъектов компьютерной системы. Решение о доступе принимается путем сопоставления уровня конфиденциальности и уровня доверия. При использовании концепции ролей задается множество разрешенных системных операций путем введения дополнительных объектов – ролей, наделенных набором разрешенных доступов. Решение о разрешении доступа принимается исходя из роли, сопоставленной субъекту.

Попытки предоставления совмещенных сервисов разграничения доступа (мандатного и ролевого) встроены в ряд систем управления базами данных. Так, например, в широко распространенной СУБД Oracle [4, стр. 54] уже в версии 7 было разработано дополнительное инструментальное средство Trusted

Copyright © 2009 **С.В. Белим, Н.Ф. Богаченко, Ю.С. Ракицкий.**
Омский государственный университет.
E-mail: sbelim@mail.ru

Oracle7, которое позволяло администратору кроме ролей вводить также и метки безопасности. Основным требованием мандатного разграничения доступа в данном приложении было доминирование метки пользователя над меткой строки. Начиная с версии СУБД Oracle8 этот продукт получил название Oracle Label Security. Однако оба эти продукта не получили широкой популярности в силу двух причин. Во-первых, согласование настроек двух алгоритмов приводит к большому количеству трудностей при администрировании. Во-вторых, остается неочевидным сама возможность непротиворечивого сосуществования двух принципов разграничения доступа в одной компьютерной системе.

Целью данной статьи ставится развитие модели ролевого разграничения доступа, а также доказательства принципиальной возможности построения политики безопасности, использующей концепцию ролей и мандатное разделение доступа. Также исследуются математические структуры, необходимые для моделирования политики безопасности.

2. Ролевая политика безопасности

Ролевая политика безопасности основывается на разрешении или запрещении действий в системе в целом без привязки к отдельным объектам системы. В общем случае такой подход реализуется с помощью концепции привилегий. Под привилегией понимается единица доступа к системной информации. Будем считать, что системная информация представима с помощью множества объектов $\mathbf{O}$. Роль – это именованная совокупность привилегий, то есть множество разрешенных типов доступа к системным объектам. Множество всех возможных типов доступа к системным объектам обозначим через $\mathbf{A}$. Перейдем к более строгому описанию модели ролевого разграничения доступа, приведенного в работе [6].

Определение 1. Под *привилегией* будем понимать пару (x, m), где x – системный объект ($x \in \mathbf{O}$), а m – непустое множество видов доступа ($m \subseteq \mathbf{A}$).

Определение 2. *Роль* – это именованное множество привилегий, которое в дальнейшем будем представлять в виде пары $(rname, rpset)$, где $rname$ – уникальный идентификатор, $rpset$ – множество привилегий.

Если определена роль r, то ее имя $r.rname$, а множество привилегий – $r.rpset$. Далее введем два множества: R – множество ролей системы, P – множество всех возможных привилегий. Также определим функцию, играющую важную роль в администрировании систем с ролевым разграничением доступа:

$$\Psi : R \to 2^{|P|}.$$

Данное отображение показывает привилегии заданной роли. По сути $\Psi(r) = r.rpset$. Через концепцию ролей осуществляется доступ к системной информации.

Пусть UID – множество идентификаторов пользователей, GID – множество идентификаторов групп пользователей, общее множество идентификаторов, для которых производится ролевое разграничение доступа $ID = UID \cup GID$.

Для систем с ролевым разграничением доступа важную роль играет процесс авторизации. Причем возможны два случая. Авторизация «Роль – Привилегия» включает заданную привилегию в множество привилегий данной роли, то есть, если роль r авторизована на привилегию p, то $p \in r.rpset$. Авторизация «Роль-Роль» подразумевает включение привилегий одной роли в множество привилегий другой роли. То есть, если роль r_1 авторизована на роль r_2, то $r_2.rpset \subseteq r_1.rpset$.

Авторизация «Роль-Роль» порождает бинарное *отношение* на множестве ролей. Обозначим это отношение через $r_1 \to r_2$, если роль r_1 авторизована на роль r_2.

Как показано в работе [6], функция Ψ монотонно возрастает по отношению к операции $\to$, то есть если $r_1 \to r_2$, то $\Psi(r_2) \subseteq \Psi(r_1)$.

Обозначим цепочку вида $r_i \to r_{i_1} \to \ldots \to r_{i_n} \to r_j$ через $r_i \to^+ r_j$ (при $n > 0$) и $r_i \to^* r_j$ (при $n \geqslant 0$).

Определение 3. *Ролевым путем* $p(r_i, r_j)$ между двумя ролями r_i и r_j будем называть цепочку $r_i \to^* r_j$.

Заданному отношению на множестве ролей можно сопоставить ориентированный граф, в котором дуга (r_1, r_2) существует тогда и только тогда, когда роль r_1 авторизована на роль r_2. Очевидно, что ролевой путь $p(r_i, r_j)$ изоморфен ориентированному пути в этом орграфе, ведущему из вершины r_i в вершину r_j.

Определение 4. *Тривиальным* является ролевой путь, состоящий из одной роли, то есть путь нулевой длины из вершины к самой себе.

Определение 5. Будем говорить, что роль r_i *доминирует* над ролью r_j, а роль r_j *подчиняется* роли r_i, если существует ролевой путь $p(r_i, r_j)$. Или в графовой постановке: вершина r_i *доминирует* над вершиной r_j, а вершина r_j *подчиняется* вершине r_i, если существует ориентированный путь $p(r_i, r_j)$.

Легко доказать, что отношение доминирования одной роли над другой задает *отношение частичного порядка* на множестве ролей R. Следует отметить, что возможно два различных случая, зависящих от принципа администрирования ролевой политики безопасности. В первом случае допускается существование ролей с совпадающим набором полномочий. Тогда отношения порядка между ролями нестрогое. Однако такой подход имеет смысл только как временная мера при формировании ролевой политики безопасности. В окончательно сформированной иерархии ролей существование двух ролей с совпадающими полномочиями лишено смысла. Второй случай исключает наличие двух ролей с совпадающими полномочиями. Для данного подхода отношение порядка будет строгим. В дальнейшем будем придерживаться именно второго подхода, обеспечивающего оптимальное управление ролями, но обозначать отношение доминирования роли r_i над ролью r_j как $r_i \geqslant r_j$ (или $r_j \leqslant r_i$).

3. Мандатная политика безопасности

Мандатные политики безопасности строятся основываясь на понятиях уровня секретности информации и уровня доверия к пользователю. Существуют различные подходы, позволяющие определять уровень секретности информации. Наиболее общий подход строится на основе решетки ценностей. Приведем основные определения из теории решеток [2, стр. 17], используемые в дальнейшем тексте.

Определение 6. *Решетка* – это частично упорядоченное множество, в котором каждое двухэлементное подмножество имеет как точную верхнюю (sup), так и точную нижнюю (inf) грани, принадлежащие этому множеству.

Определение 7. Для A, B элемент $C = \sup(A, B)$ называется *точной* или *наименьшей верхней гранью*, если:

1. $A \leqslant C$, $B \leqslant C$.
2. $\forall D : A \leqslant D,\ B \leqslant D \Rightarrow\ C \leqslant D$.

Определение 8. Для A, B элемент $E = \inf(A, B)$ называется *точной* или *наибольшей нижней гранью*, если:

1. $E \leqslant A$, $E \leqslant B$.
2. $\forall D : D \leqslant A,\ D \leqslant B \Rightarrow\ D \leqslant E$.

Каждому объекту и субъекту системы сопоставляется «метка безопасности», являющаяся элементом решетки. При запросе на доступ субъекта к объекту происходит сравнение меток безопасности. Доступ разрешен, если метка безопасности субъекта доминирует над меткой безопасности объекта, в остальных случаях доступ запрещен.

В связи с тем что основные концепции ролевого доступа сформулированы в терминах теории графов, введем аналогичные понятия для мандатного разграничения доступа.

Определение 9. *Решеточным графом* будем называть ориентированный граф[1], вершины которого образуют решетку. При этом отношение порядка задается отношением доминирования на множестве вершин графа: если $\exists\, p(r_1, r_2)$, то $r_1 \geqslant r_2$. Наименьшая верхняя грань $\sup(r_1, r_2)$ определяется как ближайшая вершина, доминирующая над r_1 и r_2. Наибольшая нижняя грань $\inf(r_1, r_2)$ определяется как ближайшая вершина, подчиненная вершинам r_1 и r_2.

Определим более формально понятия *наименьшей верхней* и *наибольшей нижней* граней в контексте ориентированного графа:
$r = \sup(r_1, r_2) \Longleftrightarrow$

[1]Если исходя из контекста понятно, что речь идет об ориентированном графе, то орграф будем называть просто графом.

1. $\exists\, p(r, r_1)$ & $p(r, r_2)$, то есть r является верхней гранью.
2. Если $\exists\, p(r', r_1)$ & $p(r', r_2)$, то $\exists\, p(r', r)$, то есть r минимальна среди всех верхних граней.

$r = \inf(r_1, r_2) \Longleftrightarrow$

1. $\exists\, p(r_1, r)$ & $p(r_2, r)$, то есть r является нижней гранью.
2. Если $\exists\, p(r_1, r')$ & $p(r_2, r')$, то $\exists\, p(r, r')$, то есть r максимальна среди всех нижних граней.

Теорема 1. *Для произвольной решетки существует изоморфный ей решеточный граф.*

Доказательство. Пусть задана некоторая произвольная решетка. Сопоставим узлам решетки вершины графа G, а отношение порядка представим ориентированными дугами, направленными от «меньшей» вершины к «большей»: $r_1 \geqslant r_2 \Longleftrightarrow \exists (r_1, r_2) \in E$, где E – множество дуг графа G.

Операции взятия inf и sup для узлов решетки r_1 и r_2 дадут те же узлы, что и в случае применения этих операций к построенному графу G (так как дуга между двумя вершинами образует путь единичной длины между ними). ■

Замечание 1. Решеточный граф, изоморфный заданной решетке, неединственен. Действительно, например, графы на рисунке 1 изоморфны одной и той же решетке (M, P), где $M = \{a, b, c, d\}$ – множество узлов решетки, $P = \{(a, b), (a, c), (a, d), (b, d), (c, d)\}$ – отношение частичного порядка, заданное на M.

Определение 10. Решеточные графы G_1 и G_2 назовем *эквивалентными* ($G_1 \sim G_2$), если они изоморфны одной и той же решетке.

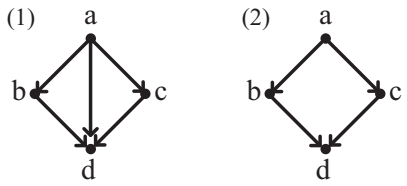

Рис. 1. Эквивалентные решеточные графы

Далее проанализируем, какими свойствами должен обладать решеточный граф и какие признаки являются достаточными условиями того, что граф решеточный.

Определение 11. Связный ориентированный граф называется *сетью*, если в нем существует единственный *источник* (вершина без входящих дуг) и единственный *сток* (вершина без исходящих дуг) [3, стр. 199].

Теорема 2. *Решеточный граф является сетью без ориентированных циклов.*

Доказательство. Пусть ориентированный граф G с множеством вершин R – решеточный.

Докажем, что в G нет ориентированных циклов. Допустим? это не так. Тогда узлы решетки, отвечающие ориентированному циклу, связаны отношением порядка: $r_i \geqslant r_{i+1} \geqslant \ldots \geqslant r_{i+n} \geqslant r_i$. По транзитивности получаем: $(r_i \geqslant r_{i+1})\&(r_{i+1} \geqslant r_i) \Rightarrow (r_i = r_{i+1})$. Но узлы решетки образуют множество, следовательно, все различны – это противоречие.

Так как граф G решеточный, то $\forall r_1, r_2 \in R\ \exists r_3 \in R : r_3 = \sup(r_1, r_2)$. По определению, $\sup(r_1, r_2)$ - это вершина, доминирующая над r_1, r_2, то есть в G существуют ориентированные пути $p(r_3, r_1)$ и $p(r_3, r_2)$. Следовательно, $\forall r_1, r_2$ существует неориентированный путь $p(r_3, r_1) \cup (r_3, r_2)$, соединяющий их, а значит, граф G связен.

Существование источника и стока следует из следующих рассуждений. Пусть, например, источника не существует, это значит, что $\forall r \in R$ найдется входящая дуга (r', r), то есть $\exists r' : r \leqslant r'$. Следовательно, $\exists \{r_i\}_{i=1}^{\infty} : r_i \leqslant r_{i+1}$. В силу конечности множества вершин $\exists i, j : (i < j)\&(r_i = r_j)$. Тогда $r_i \leqslant r_{i+1} \leqslant \ldots \leqslant r_j = r_i$. Получаем ориентированный цикл – противоречие с ранее доказанным. Существование стока доказывается аналогично.

Докажем единственность источника. Пусть это не так. Тогда существует как минимум два различных источника s_1 и s_2 ($s_1 \neq s_2$). Так как граф G решеточный, то должна существовать вершина r, из которой можно построить ориентированные пути $p(r, s_1)$ и $p(r, s_2)$. Но s_1 и s_2 – источники, следовательно, возможны лишь тривиальные пути, ведущие в эти вершины: $p(s_1, s_1)$ и $p(s_2, s_2)$. Отсюда получаем, что для существования $\sup(s_1, s_2)$ надо потребовать: $s_1 = s_2$ – это противоречие. Аналогично доказывается единственность стока. ■

Теорема 3. *Источник в решеточном графе доминирует над любой вершиной, а сток подчиняется любой вершине этого графа.*

Доказательство. Действительно, пусть s – источник. По определению решеточного графа $\forall r \in R : \exists r' = \sup(s, r)$. Следовательно, в графе найдутся ориентированные пути $p(r', r)$ и $p(r', s)$. Так как в s не входит ни одна дуга, то $r' = s$ и $\forall r \in R : \exists p(s, r)$. Аналогично доказывается, что $\forall r \in R : \exists p(r, t)$, где t – сток. ■

Утверждение, обратное к теореме 2, неверно: не любая сеть без ориентированных циклов является решеточным графом. Возможны две причины, по которым ориентированный граф не будет решеточным (частично упорядоченное множество не будет решеткой [5]):

1. Найдутся две вершины, вообще не имеющие верхней (нижней) грани.

2. Найдутся две вершины, для которых нельзя выбрать минимальную среди верхних (максимальную среди нижних) граней – они несравнимы.

Первый случай отсекается требованием существования источника и стока в орграфе и отсутствием ориентированных циклов.

Теорема 4. *Источник в сети без ориентированных циклов доминирует над любой вершиной, а сток – подчиняется любой вершине этого графа.*

Доказательство. Пусть r – произвольная вершина сети без ориентированных циклов, s – источник, t – сток.

Будем строить ориентированный путь, начиная с вершины r и добавляя на каждом шаге по одной дуге. В силу отсутствия в сети ориентированных циклов, процесс построения пути конечен, причем последней присоединенной дугой будет дуга, ведущая в единственный сток графа. Следовательно, в сети существует по крайней мере один ориентированный путь $p(r_1, t)$. Но тогда t подчиняется вершине r.

Существование ориентированного пути $p(s, r)$ доказывается аналогично, но его построение ведется в направлении, обратном ориентации дуг. Таким образом, s доминирует над вершиной r. ■

Замечание 2. Из теоремы 4 очевидным образом следует, что в сети без ориентированных циклов для любой пары вершин существуют по крайней мере одна верхняя грань (это источник) и одна нижняя грань (это сток). Но остается открытым вопрос о возможности выбора *наименьшей* верхней (*наибольшей* нижней) граней, то есть вопрос сравнимости граней.

Вторую причину «нерешеточности» графа легко проиллюстрировать примером: граф (1) на рисунке 2 – сеть без ориентированных циклов, но она не является решеточным графом. Действительно, вершины a и b имеют две несравнимые нижние грани c и d, а еще одна нижняя грань t заведомо меньше c и d; вершины c и d имеют две несравнимые верхние грани a и b, а еще одна верхняя грань s заведомо больше a и b.

Но требование отсутствия в сети подграфов, имеющих более одного стока или источника, как у подграфа, порожденного множеством вершин $\{a, b, c, d\}$ (см. рис. 2 (1)), не является достаточным условием решеточности. Действительно, добавление вершины e (см. рис. 2 (2)) делает рассмотренную сеть решеточным графом: $\inf(a, b) = e$ (c и d по-прежнему несравнимы, но $(e \geqslant c)\&(e \geqslant d)$) и $\sup(c, d) = e$ (a и b по-прежнему несравнимы, но $(e \leqslant a)\&(e \leqslant b)$).

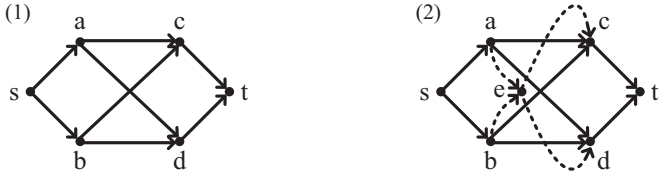

Рис. 2. Сеть, не являющаяся решеточным графом (1), и сеть, являющаяся решеточным графом (2)

Следующие теоремы дают ряд достаточных условий решеточности графа.

Теорема 5. *Пусть граф G является сетью и удаление стока превращает его в ориентированное дерево*[2]*. Тогда G – решеточный.*

Доказательство. Пусть s – источник, t – сток и $G\backslash\{t\} = T$ – дерево, полученное из исходного графа удалением стока. Если R – множество вершин графа G, то $R_T = R\backslash\{t\}$ – множество вершин дерева T и s – корень дерева.

Докажем существование наименьшей верхней грани для любой пары вершин графа G.

Пусть r_1 и r_2 ($r_1 \neq r_2$) – две произвольные вершины дерева T. По теореме о свойствах ордерева для любой его вершины существует единственный ориентированный путь, ведущий в эту вершину из корня [3, стр. 239]. Тогда $\exists\,!\; p(s, r_1)$ и $\exists\,!\; p(s, r_2)$. Так как $r_1 \neq r_2$, то эти пути не совпадают. Пусть r' – последняя, считая от s, из общих вершин этих путей. Очевидно, что все вершины, принадлежащие ориентированному пути $p(s, r')$, являются верхними гранями вершин r_1 и r_2, а r' – минимальная среди них. В силу единственности путей $p(s, r_1)$ и $p(s, r_2)$ других верхних граней у вершин r_1 и r_2 нет. Следовательно, $r' = \sup(r_1, r_2)$.

Рассмотрим теперь пару вершин (r, t), где r – произвольная вершина дерева T. По теореме 4 сток t подчиняется вершине r, то есть существует ориентированный путь $p(r, t)$. Следовательно, $\sup(r, t) = r$.

Докажем теперь существование наибольшей нижней грани для любой пары вершин графа G.

Рассмотрим сначала две произвольные вершины r_1 и r_2 ($r_1 \neq r_2$) дерева T. Возможны два случая: либо не существует ориентированного пути, связывающего эти две вершины, либо он единственен.

В первом случае, двигаясь из этих вершин по направлению дуг, построим ориентированные пути $p(r_1, \widetilde{r_1})$ и $p(r_2, \widetilde{r_2})$, где $\widetilde{r_1}$ и $\widetilde{r_2}$ – листья. Эти пути могут быть не единственны, но все они не имеют общих вершин (иначе в свободном дереве $\widetilde{T}$, полученном из T отменой ориентации ребер, был бы цикл, что противоречит теореме о свойствах ордерева [3, стр. 239]). В графе G существуют дуги $(\widetilde{r_1}, t)$ и $(\widetilde{r_2}, t)$. Следовательно, в графе G ориентированные пути $p(r_1, t) = p(r_1, \widetilde{r_1}) \cup (\widetilde{r_1}, t)$ и $p(r_2, t) = p(r_2, \widetilde{r_2}) \cup (\widetilde{r_2}, t)$ имеют лишь одну общую вершину t. Тогда $\inf(r_1, r_2) = t$.

Во втором случае существование наибольшей нижней грани очевидно, ею будет та из вершин r_1, r_2, которая является конечной в существующем между этими вершинами ориентированном пути.

Существование наибольшей нижней грани для двух вершин, одна из которых является стоком (пусть это вершины r и t), следует из существования по крайней мере одного ориентированного пути $p(r, t)$ (см. теорему 4). ■

Теорема 6. *Пусть граф G является сетью, а удаление источника и инвертирование всех дуг превращает его в дерево. Тогда G – решеточный.*

[2] В дальнейшем ориентированное дерево будем называть просто деревом, а неориентированное – *свободным* деревом.

Доказательство. Если инвертировать все дуги сети G, то мы получим сеть $\widehat{G}$, в которой источник и сток поменялись местами. Для графа $\widehat{G}$ справедлива теорема 5. Но тогда исходный граф G также будет решеточным, так как вершина r_1 доминирует над вершиной r_2 в графе $\widehat{G}$ тогда и только тогда, когда вершина r_2 доминирует над вершиной r_1 в графе G. ■

4. Древовидная иерархия ролей

Рассмотрим ситуацию, когда в компьютерной системе кроме ролевой политики безопасности необходимо реализовать также мандатную политику безопасности на основе некоторой решетки L. Основная проблема, возникающая в этом случае, состоит в построении правил доступа, удовлетворяющих обеим политикам безопасности и не противоречащих друг другу. В первую очередь рассмотрим иерархию ролей, образующую дерево.

Теорема 7. *Пусть в компьютерной системе действуют ролевая политика безопасности на основе дерева ролей T и мандатная политика безопасности на основе решетки L, тогда в компьютерной системе может быть построена непротиворечивая политика безопасности, включающая в себя разграничения обеих политик безопасности.*

Доказательство. Очевидно, что в дереве присутствует роль, доминирующая над всеми остальными ролями – корень дерева (назовем ее максимальной ролью – $MaxRole$). Добавим к дереву T вершину $MinRole$, не обладающую никакими привилегиями, и соединим дугами все листья дерева T с вершиной $MinRole$ (добавленные дуги ориентируем от листьев дерева T к $MinRole$), тем самым построим граф TM. Очевидно, что вершина $MaxRole$ является источником, вершина $MinRole$ – стоком, а граф TM – сетью. Согласно теореме 5, TM – решеточный граф, то тесть его вершины образуют решетку.

Тогда возможно получить декартово произведение решетки, построенной на вершинах графа TM, и решетки L. Как показано в [1, стр. 21], декартово произведение решеток есть решетка.

На основании полученной решетки можно построить мандатную политику безопасности. С другой стороны, из элементов решетки возможно построить решеточный граф (см. теорему 1). Такой граф может задавать ролевую политику безопасности. ■

5. Произвольная иерархия ролей

Перейдем теперь к рассмотрению иерархии ролей, образующей произвольный ориентированный граф G.

Определение 12. *Допустимым преобразованием* ориентированного графа ролей G назовем следующий процесс: если в G имеется более одного стока, то добавляется роль, не обладающая никакими привилегиями, и дуги, ведущие от стоков графа G к новой роли.

Очевидно, что, с одной стороны, такое преобразование превращают граф ролей в сеть (в случае единственности источника), а с другой – изменения ролевой политики безопасности несущественны.

Теорема 8. *Если граф иерархии ролей является решеточным либо его можно с помощью допустимого преобразования расширить до решеточного, то ролевая политика безопасности допускает непротиворечивое совмещение с мандатной политикой безопасности.*

Доказательство. Расширим, если это необходимо, граф иерархии ролей до решеточного графа, обозначим его GM. Мандатная политика безопасности задается решеткой L. Тогда можно взять декартово произведение решетки, построенной на вершинах решеточного графа GM, и решетки L. Согласно [1, стр. 21], такое декартово произведение, обозначим его через $GM \times L$, является решеткой.

Поскольку $GM \times L$ – решетка, то она может задавать мандатную политику безопасности. С другой стороны, на основании решетки $GM \times L$ можно построить решеточный граф (см. теорему 1), который будет задавать ролевую политику безопасности. ■

6. Пример совмещения ролевой и мандатной политик безопасности

Доказательство теоремы 8 является конструктивным. В качестве иллюстрации предложенного алгоритма объединим ролевую политику безопасности, представленную на рисунке 3, и мандатную политику безопасности, построенную на линейном множестве из трех элементов.

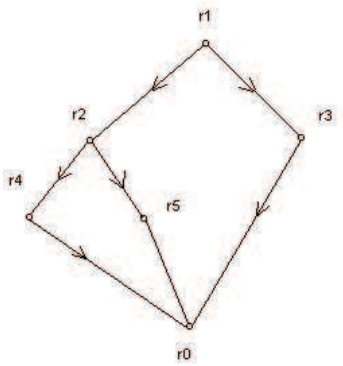

Рис. 3. Ролевая политика безопасности

Ролевая политика задается шестью ролями, одна из которых (r_0) является «пустой», то есть не обладающей какими-либо привилегиями и подчиненной любой другой роли. Очевидно, что заданная политика безопасности соответствует условиям теоремы 5. Следовательно, граф, представленный на рисунке 3, является решеточным.

Пусть мандатная политика безопасности задается решеткой L, элементами которой являются узлы l_1, l_2, l_3, причем отношение порядка задано таким образом, что $l_1 \geqslant l_2 \geqslant l_3$.

Согласно теореме 8, возможно непротиворечивое совмещение заданных политик безопасности. Для этого необходимо построить решетку $R \times L$, являющуюся декартовым произведением решеток R и L, где R – решетка, определяемая решеточным графом, представленным на рисунке 3.

Элементами решетки $L \times R$ являются пары (r_i, l_j), при $i = 0, \ldots, 5$ и $j = 1, \ldots, 3$. При этом отношение порядка задается следующим образом: $(r_i, l_j) \geqslant (r_k, l_m)$, если $r_i \geqslant r_k$ и $l_j \geqslant l_m$. Заметим, что узлы r_2 и r_3, r_4 и r_5, r_3 и r_4, r_3 и r_5 попарно несравнимы. Решеточный граф, изоморфный решетке $R \times L$, представлен на рисунке 4.

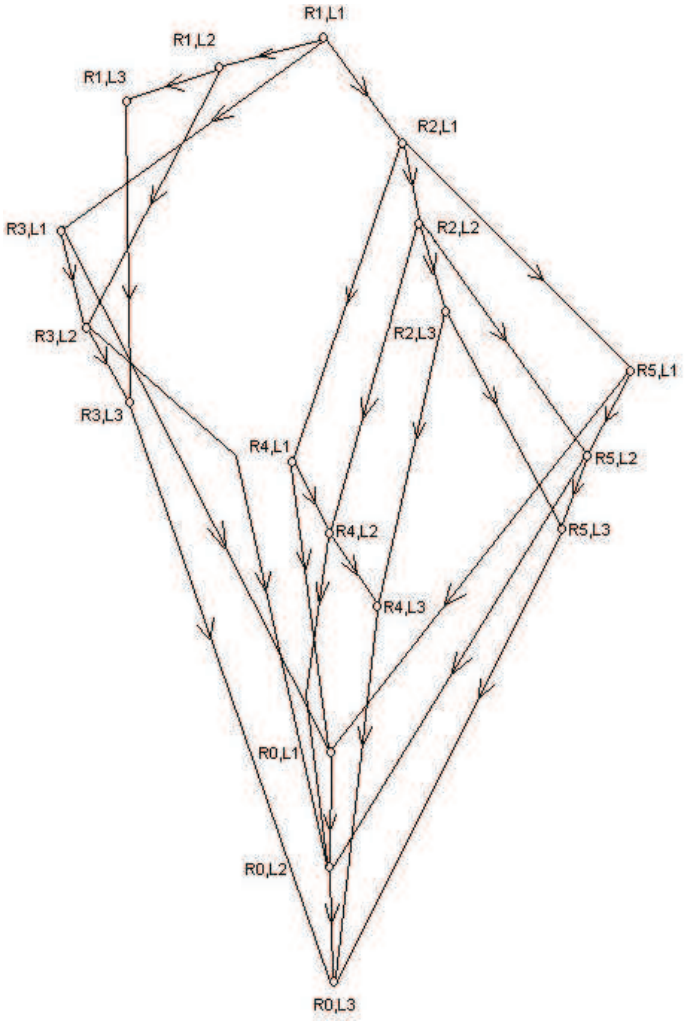

Рис. 4. Совмещение ролевой и мандатной политик безопасности

На полученной решетке $R \times L$ можно задать мандатную политику безопасности. В свою очередь, на полученном ориентированном графе (см. рис. 4) можно построить ролевую политику безопасности.

7. Заключение

Таким образом, возможно создание политики безопасности предприятия, включающей в себя мандатное и ролевое разграничение доступа. Причем результат объединения этих двух подходов может быть представлен как в виде концепции, основанной на метках безопасности, так и в виде иерархии ролей.

Литература

1. Биркгоф, Г. Теория решеток / Г. Биркгоф. – М.: Наука. Главная редакция физико-математической литературы, 1984. – 568 с.
2. Гретцер, Г. Общая теория решеток / Г. Гретцер. / Под редакцией Д.М. Смирнова. – М.: Мир, 1981. – 456 с.
3. Новиков, Ф.А. Дискретная математика для программистов / Ф.А. Новиков. – СПб.: Питер, 2001. – 304 с.
4. Терьо, М. Oracle. Руководство по безопасности / М. Терьо, А. Ньюмен. – М.: Издательство «ЛОРИ», 2004. – 560 с.
5. Решетки [Электронный ресурс]. – Режим доступа: http://www.eltech.ru/misc/LGA_2007_FINAL/Allpage/Section8/Part8112.htm (09.10.2009).
6. Nyanchama, M. Access Rights Administration in Role-Based Security Systems / M. Nyanchama, S.L. Osborn // Database Security VIII: Status & Prospects, Biskup, Morgenstern and Landwehr, eds. North-Holland. – 1994. – C. 37-56.

Математические структуры и моделирование
2009, вып. 20, с. 153–159

УДК 65.012.810

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД В ПОСТРОЕНИИ ДИСКРЕЦИОННОЙ ПОЛИТИКИ БЕЗОПАСНОСТИ

С.В. Белим, С.В. Усов

Проведена модификация модели дискреционного разделения доступа HRU для объектно-ориентированных компьютерных систем.

Введение

Традиционно рассматриваемая модель системы безопасности компьютерных систем строится на основе субъектно-объектного подхода. Однако в последнее время, в связи с распространением объектно-ориентированного подхода в построении программного обеспечения компьютерных систем, складывается двойственная ситуация, когда один и тот же набор данных в одном случае интерпретируется как объект (пассивный), в другом случае как субъект (активный). В связи с этим, для более адекватного описания компьютерных систем, необходимо применение объектно-ориентированного подхода и соответствующая модификация моделей политик безопасности. В частности, объектно-ориентированная модель компьютерной системы построена в работе [5], где даны и необходимые определения.

В случае объектно-ориентированной модели компьютерной системы, как и для субъектно-объектного подхода, возможно определение элементарных операторов в рамках модели безопасности HRU [3]. Поскольку объектно-ориентированный подход предполагает разбиение объектов на классы, задаваемые списком полей и методов, то при построении модели HRU мы должны научиться обращаться не только с представителями классов, но и с самими классами. Во-первых, рассматривая классы в качестве реализации типов из модели типизированной матрицы доступов (TAM), построенной на основе HRU, мы получаем менее жесткие условия в критерии безопасности системы [6]. Во-вторых, возникает два варианта определения как локальных (привязанных к каждому объекту) матриц доступа, так и элементарных операторов, преобразующих эти матрицы.

Первый подход, классово-типизированный, отождествляет класс объекта с его типом, и матрица доступа к каждому объекту каждого класса строится как

Copyright © 2009 **С.В. Белим, С.В. Усов.**
Омский государственный университет им. Ф.М. Достоевского.
E-mail: belim@univer.omsk.su

ТАМ с учетом объектно-ориентированного подхода к строению компьютерной системы.

Второй подход, наследственно-типизированный, обращается к наследственной структуре объектов в объектно-ориентированной среде. В этом случае все объекты одного класса, а значит, одного типа, являются эквивалентными с точки зрения набора прав доступа, и матрицу доступа достаточно задать для класса целиком. С другой стороны, множество типов является частично упорядоченным, причем порядок задается наследственностью соответствующих типам классов. Более того, наследственность проявляется и в разграничении прав доступа: объект класса-наследника обладает всеми правами класса-родителя. Также имеет смысл ввести запрет на доступ к полю (методу) класса-наследника для объекта, у которого нет права доступа к соответствующему полю (методу) класса-родителя.

Однако в данной работе мы ограничимся построением объектно-ориентированной модели системы безопасности в рамках классово-типизированного подхода и выявлением случаев, допускающих проверку безопасности компьютерной системы.

1. Элементарные операторы модели HRU в объектно - ориентированной компьютерной системе

Будем рассматривать компьютерную систему в виде множества объектов O, имеющих открытые поля f и скрытые поля p, а также методы обработки полей s.

Все необходимые определения даны в работе [5], там же построена и модель системы безопасности для объектно-ориентированной компьютерной системы.

Матрица доступа M в данном случае - скрытое (private) поле объекта. От объекта к объекту даже одного класса (типа) содержание этого поля может отличаться. Строки матрицы объекта o_i соответствуют объектам компьютерной системы, столбцы – открытым (public) полям и методам объекта o_i, элементом матрицы является некоторое подмножество множества R видов доступа к полю, если столбец элемента соответствует public полю объекта, и принимает значение 0 либо 1, если столбец элемента соответствует методу объекта, где "1" разрешает вызов метода, а "0" - запрещает.

Определим следующие элементарные операции для работы с матрицей доступов:

1) $Create(o_j, \tau)$ – создает объект o_i класса $\tau \in \mathbf{T}$, при этом $O' = O \cup \{o_j\}$, а в матрице доступа объектов o_i $(i \neq j)$ добавляется строка, соответствующая объекту o_j, причем $\forall f \in o_i.F$ $o_i.M'[o_j, o_i.f] = \emptyset$, $\forall s \in o_i.S$ $o_i.M'[o_j, o_i.s] = \emptyset$. Матрица доступа объекта o_j создается полностью пустой. 2) $Destroy(o_j)$ – уничтожает объект o_j, при этом $O' = O \diagdown \{o_j\}$, а в матрице доступа объектов o_i $(i \neq j)$ уничтожается строка, соответствующая объекту o_j. 3) $Enter(r, o_j, o_i.f)$ - вносит право доступа r в $o_i.M[o_j, o_i.f]$, при этом матрица доступа изменяется по правилу: $o_i.M'[o_j, o_i.f] = o_i.M[o_j, o_i.f] \cup \{r\}$. 4) $Delete(r, o_j, o_i.f)$ – удаляет право r доступа из $o_i.M[o_j, o_i.f]$, при этом матрица доступа изменяется по

правилу: $o_i.M'[o_j, o_i.f] = o_i.M[o_j, o_i.f] \setminus \{r\}$. 5) $Grant(o_j, o_i.s)$ – разрешает вызов объекту o_j метода $o_i.s$, при этом матрица доступа изменяется по правилу: $o_i.M'[o_j, o_i.s] = 1$. 6) $Deprive(o_j, o_i.s)$ – запрещает вызов объекту o_j метода $o_i.s$, при этом матрица доступа изменяется по правилу: $o_i.M'[o_j, o_i.s] = \emptyset$.

В дальнейшем будем сокращать запись $o_i.M[o_j, o_i.f]$ до $M_i[j, f]$ и запись $o_i.M[o_j, o_i.s]$ до $M_i[j, s]$.

2. Определения

Определение 1. Под состоянием компьютерной системы в момент времени t будем понимать пару $Q(t) = (O(t), M(t))$, где $O(t) = \{o_j\}$ – множество всех объектов системы в момент времени t, а $M(t) = \{o_j.M[](t)\}$ – семейство всех матриц доступа объектов системы в состоянии на момент t. Начальное состояние системы будем обозначать $Q(0)$.

Состояния компьютерной системы в модели HRU изменяются под воздействием запросов на модификацию матрицы доступа в виде команд следующего формата:

Команда $\gamma(x_1, \ldots, x_\gamma)$:

if — условная часть (представляет собой конъюнкцию значений логических выражений вида $r \in M_i[j, f]$ или $M_i[j, s] = 1$)

$then$ — последовательность элементарных операций.

Здесь аргументы x_k представляют собой либо объекты, либо поля или функции объектов, участвующие в качестве аргументов в условной части либо в элементарных операциях.

Факт перехода системы под действием команды γ из состояния $Q(t)$, в котором она находилась в момент времени t, в состояние $Q(t+1)$, в котором она находилась в следующий момент времени $t+1$, будем записывать в сокращенном виде: $Q(t) \to_\gamma Q(t+1)$.

Определение 2. HRU-модель системы безопасности называется монооперационной, если каждая команда в этой системе содержит только одну элементарную операцию.

Определение 3. HRU-модель системы безопасности называется монотонной, если каждая команда в этой системе содержит только одно условие.

Определение 4. HRU-модель системы безопасности объектно-ориентированной компьютерной системы называется монотонной, если команды этой системы не содержат операций *Delete*, *Deprive* и *Destroy*.

Определение 5. Будем говорить, что состояние системы $Q(t)$ допускает утечку права доступа $r \in R$, если существуют такие команда γ, объект o_j и поле $o_i.f$ объекта o_i, что $r \notin M_i[j, f](t)$, но $r \in M_i[j, f](t+1)$, где $Q(t) \to_\gamma Q(t+1)$.

Определение 6. Будем говорить, что состояние системы $Q(t)$ допускает утечку права вызова, если существуют такая команда γ, объект o_j и метод $o_i.s$ объекта o_i, что $M_i[j, f](t) = 0$, но $M_i[j, f](t+1) = 1$, где $Q(t) \to_\gamma Q(t+1)$.

Определение 7. Будем говорить, что система r-безопасна, если не существует такой последовательности команд $\gamma_1 \dots \gamma_T$, что $Q(t-1) \rightarrow_{\gamma_t} Q(t)$, $t = 1, 2 \dots T$, и $Q(T)$ допускает утечку права r либо утечку права вызова.

3. Случаи, допускающие проверку безопасности модели HRU объектно-ориентированной компьютерной системы

Попытаемся ответить на вопрос, в каких случаях возможно решить проблему r-безопасности системы для объектно-ориентированной системы. Разрешимость аналогичной проблемы для HRU-моделей субъектно-объектных систем уже была доказана для некоторых частных случаев. А именно в [4] - для монооперационных систем и в [2] – для монотонных моноусловных систем. Наша цель – повторить эти результаты в рамках HRU-модели объектно-ориентированной системы.

Теорема 1. *Проблема r-безопасности системы является NP-разрешимой для монооперационной объектно-ориентированной модели.*

Доказательство. Если мы покажем, что для разрешения проблемы r-безопасности достаточно проверить конечное число последовательностей команд конечной длины, теорема будет доказана.

Без ограничения общности считаем, что нам достаточно обнаружить только утечку права доступа к полю объекта, поскольку право вызова метода объекта можно интерпретировать как особое и единственное право доступа к этому методу.

Поскольку система монооперационна, мы можем называть команду по единственной элементарной операции, которую она содержит.

Будем искать цепочку команд наименьшей длины, переводящую систему в состояние, допускающее утечку некоторого права r. Заметим, что такая цепочка не должна содержать команд, не изменяющих ни одну матрицу доступа (как, например, команда, добавляющая право r в ячейку матрицы доступа, в которой право r уже имеется).

Допустим,

$$Q(0) \rightarrow_{\gamma_1} Q(1) \rightarrow \dots \rightarrow_{T-1} Q(T-1) \rightarrow_{\gamma_T} Q(T),$$

где $Q(T)$ допускает утечку права $r \in R$ в ячейку $M_i[j, f](T)$.

Обратим внимание, что условия команд проверяют лишь наличие права в ячейке матрицы доступа, а не его отсутствие. Таким образом, если мы выкинем все команды Delete и Destroy из цепочки, все права доступа, находившиеся в определенной ячейке в состоянии $Q(T)$, останутся там и после удаления указанных команд. Возможно, появятся новые права – и это может привести к появлению состояния, допускающего утечку права r в момент $t < T$, что только укоротит цепочку.

Однако, если для некоторого $t < T$ право r лежало в $M_i[i,f](t)$, удаление команд $Delete(r, o_j, o_i.f)$, $Destroy(o_j)$, $Destroy(o_i)$, выполненных после момента t, приведет к тому что право r останется в $M_i[j,f](T)$ и состояние $Q(T)$ более не будет допускать утечки этого права. В этом случае хотя бы одну из перечисленных команд необходимо оставить в цепочке.

Отыщем первую команду $Create$ в рассматриваемой цепочке.

(а) Такой команды нет. Значит, все команды Destroy можно удалить, поскольку среди них отсутствуют команды $Destroy(o_j)$, $Destroy(o_i)$: если бы таковые присутствовали, ячейки $M_i[i,f](T)$ просто бы не существовало. Возможно, в цепочке присутствует команда $Delete(r, o_j, o_i.f)$. Все остальные команды - это команды $Enter$, и их не более $|R||O|^2N$. Здесь N – максимальное количество полей и методов в классе объектно-ориентированной модели. Заметим, что эту оценку можно улучшить, если различать в доказательстве поля и методы класса до следующей: $(|R|N_f + N_s)|O|^2$, где N_f – максимальное количество полей, а N_s - максимальное количество методов в классе объектно-ориентированной модели. Действительно, в матрицу доступа объекта не удастся внести более $(|R|N_f + N_s)|O|$ прав, а команды, не изменяющие матрицы доступа, мы из цепочки выкидываем.

(б) Нашлась команда $Create(o_k, \tau_k)$, $\tau_k \in \mathbf{T}$. Если $o_k \notin O(0)$, то перед ней нет команды $Destroy(o_k)$, а после нее все такие команды можно удалить.

Если $o_k \in O(0)$, k отлично от i и j, то перед командой $Create(o_k, \tau_k)$ найдется ровно одна команда $Destroy(o_k)$, и мы можем удалить эту пару команд (тем самым сократив цепочку, переводящую систему в состояние, допускающее утечку права), при этом все права в матрице доступов сохранятся, и к ним добавятся новые, ранее удаленные со строкой объекта o_k. После команды $Create(o_k, \tau_k)$ все команды $Destroy(o_k)$ также можно удалить.

Наконец, $o_k \in O(0)$, $k = i$ или $k = j$. По соображениям, изложенным выше, одну пару команд вида $Destroy(o_k)$ - $Create(o_k, \tau_k)$ (если их несколько, то первую) придется оставить. Остальные из цепочки можно удалить: действительно, если в момент выполнения t команды $Destroy(o_l)$, где $l = i$ или $l = j$, $r \in R$ лежит в $M_i[j,f](t)$, то утечка права уже произошла (и цепочку можно сократить), если же $r \notin M_i[j,f](t)$, удаление пары $Destroy(o_l)$ - $Create(o_l, \tau_l)$ не вызовет появления права $r \in R$ в $M_i[j,f]$ вплоть до момента t_1 выполнения команды $Create(o_l, \tau_l)$. По аналогичной причине команду $Delete(r, o_j, o_i.f)$ можно удалить так же.

Далее, не имеет смысла оставлять более одной команды вида $Create(o_k, \tau_k)$ для каждого класса $\tau_k \in \mathbf{T}$. Действительно, второй созданный класс o_l того же типа τ_k можно отождествить с первым, o_k, выкинув из цепочки операцию $Create(o_l, \tau_k)$ и заменив далее в аргументах команд цепочки o_l на o_k. Такое отождествление остается в силе, даже если o_l ранее был разрушен, а теперь воссоздан, а l совпадает с j (или i): действительно, созданная с нуля строка для o_j (матрица доступа для o_i) не может содержать права доступа, отсутствующие в соответствующей строке M_k (матрице M_k).

Итак, в цепочке осталось не более $|\mathbf{T}|$ команд $Create$, не более одной команды $Destroy$ или $Delete$ и конечное число команд $Enter$, а именно не более

$|R|(|O|+|\mathbf{T}|+1)^2N + |R|(|O|+|\mathbf{T}|)N$. Здесь второе слагаемое отвечает за заполнение строк, соответствующих ранее удаленному, а теперь воссозданному объекту o_i (или o_j).

Таким образом, теорема доказана. ■

Теорема 2. *Проблема r-безопасности системы является NP-разрешимой для моноусловной монотонной объектно-ориентированной модели.*

Доказательство. Если мы покажем, что для разрешения проблемы r-безопасности достаточно проверить конечное число последовательностей команд конечной длины, теорема будет доказана.

Без ограничения общности считаем, что нам достаточно обнаружить только утечку права доступа к полю объекта, поскольку право вызова метода объекта можно интерпретировать как особое и единственное право доступа к этому методу.

Будем искать цепочку команд наименьшей длины, переводящую систему в состояние, допускающее утечку некоторого права r. Заметим, что такая цепочка не должна содержать команд, не изменяющих ни одну матрицу доступа (как, например, команда, добавляющая право r в ячейку матрицы доступа, в которой право r уже имеется).

Допустим,

$$Q(0) \to_{\gamma_1} Q(1) \to \cdots \to_{T-1} Q(T-1) \to_{\gamma_T} Q(T),$$

где $Q(T-1)$ допускает утечку права $r \in R$ в ячейку $M_i[j,f](T-1)$ посредством выполнения команды γ_T.

Условие выполнения команды γ_T требует наличия права доступа r_1 в ячейке $M_{i_1}[j_1,f_1](T-1)$, которое добавлено туда некоторой командой γ_t, $t<T$. Таким образом, команды $\gamma_{t+1}\dots\gamma_{T-1}$ можно выкинуть из цепочки, поскольку они заполняют ячейки, содержание которых нам не потребуется для утечки права r. Оставим лишь команду (или две), содержащую элементарную операцию $Create(o_k,\tau_k)$ (k может совпадать с i или с j), если она встретилась на выкидываемом отрезке цепочки. Далее условие команды γ_t требует наличия права доступа r_2 в ячейке $M_{i_2}[j_2,f_2](t-1)$, которое добавлено туда некоторой командой γ'_t, $t'<t$, и мы вновь можем отказаться от промежуточных команд, кроме одной-двух $Create$. Действуя таким образом, доберемся наконец до команды, условие которой выполнялось уже в начальном состоянии системы. Далее считаем, из цепочки

$$Q(0) \to_{\gamma_1} Q(1) \to \cdots \to_{T-1} Q(T-1) \to_{\gamma_T} Q(T)$$

уже выкинуты лишние команды.

Теперь добьемся того, чтобы в условии команд цепочки встречалось не более одного нового (то есть отсутствовавшего в начальном состоянии системы) объекта каждого типа. Допустим, первым был создан объект o_k типа $\tau_k \in \mathbf{T}$, а аргументом некоторых команд γ_t является объект o_l того же типа, созданный позже. Подставив вместо него o_k, получаем команды γ'_t на месте γ_t в цепочке,

причем все права, находившиеся в строчках o_l матриц доступа (и в матрице доступа объекта o_l) теперь находятся в строчках o_k матриц доступа (и в матрице доступа объекта o_k). Учитывая, что там же находятся и права доступа самого объекта o_k, утечка права доступа произойдет в момент T или даже раньше.

Итак, каждая из команд цепочки либо создает один из не более чем $\mathbf{T}$ объектов, которые встретятся в условиях команд цепочки, либо заносит право в ячейку матрицы доступа уже существующего объекта, причем наличие этого права также потребуется в условии одной из команд цепочки. На самом деле команда может выполнять сразу несколько из этих «полезных» элементарных операций, что приведет лишь к сокращению длины цепочки. В любом случае, ее длина не превышает $|R|(|O| + |\mathbf{T}|)^2 N + |\mathbf{T}|$. ■

Теорема 3. *Проблема r-безопасности системы неразрешима для произвольной объектно-ориентированной модели.*

Доказательство данной теоремы практически не отличается от классического доказательства со сведением к проблеме остановки машины Тьюринга, предложенного в работе [3]. Отличие заключается лишь в том, что лента машины Тьюринга собирается из ячеек не одной матрицы доступа, как было в случае стандартной HRU-модели, а нескольких.

Литература

1. Harrison, M.A. Theoretical issues concerning protection in operating systems / M.A. Harrison. – University of California at Berkeley, CSD-84-170, 1984.
2. Harrison, M.A. Monotonic protection systems / M.A. Harrison, W.L. Ruzzo / In R.A. DeMillo, D.P. Dobkin, A.K. Jones, R.J. Lipton, editors. – Foundations of Secure Computation. – Academic Press, Inc., 1978. – P. 461–471.
3. Harrison, M.A. Protection in operating systems / M.A. Harrison, W.L. Ruzzo, J.D. Ullman. – Communications of the ACM, 19(8):461–471, August 1976.
4. Tripunitara, M.V. The Foundational work of Harrison-Ruzzo-Ullman Revisited / M.V. Tripunitara, N. Li. – 2006.
5. Белим, С.В. Объектно-ориентированная модель компьютерной системы / С.В. Белим, С.Ю. Белим // Математические структуры и моделирование. – 2008. – Вып. 18. – С. 98-101.
6. Sandhu, R.S. The Typed Access Matrix Model / R.S. Sandhu // Proceedings of IEEE Symposium on Security and Privacy, Oakland, California, May 4-6, 1992. – P. 122-136.

Математические структуры и моделирование
2009, вып. 20, с. 160–172

УДК 65.012.810(075.8)

АЛГОРИТМЫ АНАЛИЗА БЕЗОПАСНОСТИ СОСТОЯНИЙ КОМПЬЮТЕРНОЙ СИСТЕМЫ ДЛЯ МОДЕЛИ TAKE-GRANT

Д.М. Бречка

В статье рассматриваются алгоритмы поиска tg-путей, островов и мостов в графах доступов для дискреционной модели Take-Grant.

Введение

Современные системы защиты информации создаются и тестируются исходя из постулата, впервые предложенного в знаменитой «Оранжевой книге»: «Все вопросы безопасности компьютерных систем могут быть разрешены в терминах доступов субъектов к объектам». Таким образом, основной целью любой системы защиты является разграничение доступа, позволяющего разделить компоненты компьютерной системы таким образом, что пользователи системы получают доступ лишь к той информации, которая необходима им для выполнения своих функциональных обязанностей.

При моделировании систем с дискреционным разграничением доступа строится система, отслеживающая все возможные доступы субъектов компьютерной системы к объектам компьютерной системы. Одним из представлений множества возможных доступов является граф доступов. Наиболее развитой моделью распространения прав доступа по графу доступов является Take-Grant [1]. Центральным понятием этой модели служит «безопасное состояние». В рамках модели Take-Grant выделены критерии безопасности состояния компьютерной системы. Однако остается открытым вопрос об алгоритмах проверки безопасности конкретного состояния. В данной статье предложены некоторые алгоритмы, позволяющие однозначно ответить, является ли заданное состояние компьютерной системы безопасным.

1. Описание модели Take-Grant

Модель Take-Grant — это модель дискреционного разграничения доступа в информационной системе. Данная модель является расширением классической модели HRU, ориентированной на анализ распространения прав в системе [1, 2].

Copyright © 2009 **Д.М. Бречка.**
Омский государственный университет имени Ф.М. Достоевского.
E-mail: dbrechkawork@yandex.ru

1.1. Основные положения

Согласно модели Take-Grant, информационная система представляется в виде ориентированного графа, в котором вершинами являются субъекты и объекты (активные и пассивные сущности системы соответственно), а дуги представляют собой установленные права субъектов на объекты. При этом субъекты рассматриваются как активизированные объекты, то есть могут существовать дуги между субъектами. Помимо прав, присутствующих в модели HRU, в Take-Grant добавляются два дополнительных права: Take (t) — право брать права доступа у какого-либо субъекта и Grant (g) — право предоставлять права доступа на какой-либо объект.

Состояния системы изменяются под воздействием четырех видов команд:

1. $Take(\alpha, x, y, z)$ — субъект x берет права α у объекта y на объект z.
2. $Grant(\alpha, x, y, z)$ — субъект x дает права α объекту y на объект z.
3. $Create(\alpha, x, y)$ — субъект x создает объект y с правами α на него.
4. $Remove(\alpha, x, y)$ — субъект x удаляет права α на объект y.

При анализе безопасности системы исследуется возможность получения доступа каким-либо субъектом на какой-либо объект. При этом анализируются установленные на начальный момент времени отношения в системе и рассматриваются возможности перехода системы в различные состояния путем изменения прав доступа на объекты. Имеется два возможных варианта получения прав доступа в системе: санкционированное получение прав и похищение прав.

1.2. Получение прав доступа

Определение 1. Предикат *«возможен доступ»*(α, x, y, G_0) истинен тогда и только тогда, когда существуют графы доступов G_1, G_2, ...G_n такие, что $G_1 \vdash_{c_1} G_2 \vdash_{c_2} ... \vdash_{c_n} G_n$, где G_0 — исходный граф системы, G_1, G_2, ...G_n — последовательные преобразования исходного графа под воздействием команд c_1, c_2, ...c_n.

Определение 2. Вершины графа доступов называются tg-связными (соединены tg-путем), если в графе существует такой путь между этими вершинами, что каждая дуга пути содержит право Take либо Grant без учета направления дуг.

Для графа доступов, вершинами которого являются только субъекты, справедлива следующая теорема:

Теорема 1. *В графе доступов G_0, содержащим только вершины-субъекты, предикат* «возможен доступ»(α, x, y, G_0) *истинен тогда и только тогда, когда выполнены следующие условия:*

1. *В графе G_0 существуют субъекты $s_1, s_2, ..., s_m$, связанные дугами с вершиной y.*

2. *Субъект x в графе G_0 соединен tg-путем с каждым субъектом s_i для $i = 1, 2, \ldots m$.*

Доказательство приводится в [1].

Подобная теорема существует и для произвольного графа доступов. Для того чтобы сформулировать теорему, необходимо прежде определить несколько понятий.

Определение 3. Островом в произвольном графе доступов называется его максимальный tg-связный подграф, состоящий только из вершин субъектов.

Определение 4. Мостом в произвольном графе доступов называется tg-путь, проходящий через вершины-объекты, концами которого являются вершины-субъекты; при этом словарная запись tg-пути должна иметь вид $\overrightarrow{t}^*$, $\overleftarrow{t}^*$, $\overrightarrow{t}^*\overrightarrow{g}\overleftarrow{t}^*$, $\overrightarrow{t}^*\overleftarrow{g}\overleftarrow{t}^*$, где символ * означает многократное (в том числе нулевое) повторение.

Определение 5. Начальным пролетом моста в произвольном графе доступов называется tg-путь, проходящий через вершины-объекты, началом которого является вершина-субъект, а концом — вершина-объект; при этом словарная запись пути имеет вид $\overrightarrow{t}^*\overrightarrow{g}$.

Определение 6. Конечным пролетом моста в произвольном графе доступов называется tg-путь, проходящий через вершины-объекты, началом которого является вершина-субъект, а концом — вершина-объект; при этом словарная запись пути имеет вид $\overrightarrow{t}^*$.

Теорема 2. *В произвольном графе доступов для объектов x и y предикат* «возможен доступ»(α, x, y, G_0) *истинен тогда и только тогда, когда выполнены следующие условия:*

1. *В исходном графе G_0 существуют объекты $s_1, s_2, ..., s_m$, соединенные дугами с объектом y.*

2. *В исходном графе G_0 существуют субъекты $x'_1, x'_2, ..., x'_m$ и $s'_1, s'_2, ..., s'_m$ такие, что:*

   - *$x'_i = x$ или x'_i соединен с x начальным пролетом моста в графе G_0 для $i = 1, 2, ..., m$;*
   - *$s'_i = s$ или s'_i соединен с s конечным пролетом моста в графе G_0 для $i = 1, 2, ..., m$.*

3. *В исходном графе G_0 для каждой пары (x'_i, s'_i), где $i = 1, 2, ..., m$, существуют острова $I_{i,1}, ..., I_{i,u_i}$ при $u_i \geq 1$ такие, что $x'_i \in I_{i,1}$, $s'_i \in I_{i,u_i}$, и существуют мосты между островами I_{ij} и I_{ij+1} при $j = 1, 2, ..., u_{i-1}$.*

Доказательство приводится в [1].

2. Граф доступов из вершин-субъектов

Согласно Теореме 1 для того, чтобы выяснить возможность доступа одного субъекта к другому в графе, который содержит только вершины-субъекты, необходимо выяснить, существует ли в графе между этими субъектами tg-путь. При этом направление дуг не учитывается. Для поиска tg-пути можно использовать алгоритм Дейкстры [3] при некоторой модификации графа доступов.

3. Поиск tg-путей

Для начала приведем описание самого алгоритма Дейкстры. Алгоритм позволяет найти кратчайший путь в графе между двумя выделенными вершинами s и t. Перед началом работы алгоритма все вершины и дуги не помечены. В ходе выполнения каждой вершине x присваивается число $d(x)$, равное длине кратчайшего пути между s и x, включающего только помеченные вершины и дуги. Шаги алгоритма:

1. Присвоить $d(s) = 0$ и $d(x) = \infty$ для всех x, отличных от s. Пометить вершину s и присвоить $y = s$. Где y - последняя помеченная вершина.

2. Для каждой непомеченной вершины x пересчитать величину $d(x)$:

   $d(x) = min\{d(x), d(y) + a(y, x)\}$, где $a(y, x)$ — длина дуги от y к x.

   Если $d(x) = \infty$ для всех непомеченных x, то закончить процедуру: в графе нет пути из s в непомеченные вершины. Иначе — пометить ту из вершин x, для которой $d(x)$ минимальна. Вместе с вершиной x окрашивается дуга, ведущая в данную вершину. Присвоить $y = x$.

3. Если $y == t$, то закончить процедуру: путь из s в t найден — состоит из помеченных дуг и вершин. Иначе — перейти на шаг 2.

Для того, чтобы можно было применить данный алгоритм для поиска tg-путей, произведем модификацию графа доступов описанным ниже образом.

Во-первых, будем считать, что направления дуг не имеют значения (согласно определению 2 направления дуг в tg-пути не учитываются).

Во-вторых, присвоим всем дугам, не содержащим прав Take или Grant, вес, равный бесконечности ($d(e) = \infty$).

В-третьих, для всех ребер, содержащих права Take или Grant, присвоим вес, равный единице ($d(e) = 1$), так как не существует принципиальной разницы, используется право Take или Grant. При этом если между двумя вершинами существует больше одного такого ребра, то необходимо оставить лишь одно из них.

После такой модификации к графу можно применять алгоритм Дейкстры. При этом будет найден кратчайший tg-путь между указанными вершинами, но для нас нет принципиальной разницы — любой из существующих путей подойдет.

4. Произвольный граф доступов

Согласно Теореме 2 для того, чтобы выяснить возможность доступа одного субъекта к другому в произвольном графе доступов, необходимо, чтобы:

1) в графе существовали мосты;

2) исследуемые субъекты были соединены в графе доступов начальным и конечным пролетом моста с островами;

3) сами острова в графе были соединены мостами.

4.1. Поиск островов

Для поиска островов в графе можно воспользоваться алгоритмом Флойда [3], при этом граф доступов необходимо определенным образом модифицировать.

Приведем описание алгоритма Флойда. Алгоритм предполагает, что перед началом его работы все вершины графа должны быть пронумерованы. Длина кратчайшего пути из вершины i в вершину j обозначается через d_{ij}^m, где m — число промежуточных вершин пути. Если между вершинами i и j не существует ни одного пути, то $d_{ij}^m = \infty$. Величина $d_{ii}^0 = 0$.

Формально алгоритм Флойда имеет следующие шаги:

1. Пронумеровать все вершины исходного графа целыми числами от 1 до N. Определить матрицу D^0 такую, что каждый ее элемент (i, j) соответствует величине d_{ij}^0, то есть, каждый элемент матрицы D^0 выражает длину дуги между вершинами i и j в исходном графе. Если в исходном графе вершины не соединены дугами, то $d_{ij}^0 = \infty$. Для каждого i $d_{ii}^0 = 0$.

2. Для m, последовательно принимающего значения $1, 2, ..., N$, определить по величинам элементов матрицы D^{m-1} величины матрицы D^m, используя отношение $d_{ij}^m = min\{d_{im}^{m-1} + d_{mj}^{m-1}, d_{ij}^{m-1}\}$.

По окончании данной процедуры величина элемента (i, j) матрицы D^N будет определять длину кратчайшего пути из вершины i в вершину j.

Для применения алгоритма Флойда необходимо модифицировать граф доступов таким же образом, что и для применения алгоритма Дейкстры.

1. Удалим направления дуг.

2. Присвоим всем ребрам, не содержащим права Take и Grant, бесконечный вес. Заметим, что по условию алгоритма вес ребра между двумя вершинами равен бесконечности, если ребра между вершинами не существует. Фактически это означает, что мы удаляем из графа все ребра, не содержащие прав Take или Grant.

3. Всем ребрам, содержащим права Take и Grant, присвоим вес, равный единице.

Для модифицированного графа будем применять алгоритм Флойда следующим образом:

1. Выберем вершину из множества всех вершин графа (E), применим для нее алгоритм Флойда. Алгоритм обнаружит все вершины в графе, связанные с выбранной tg-путем. Множество вершин, выбранное алгоритмом, будет являться островом. Обозначим его через I_1.

2. Множество вершин графа без I_1 обозначим через E/I_1. Выберем вершину из множества E/I_1 и применим для нее алгоритм Флойда. Алгоритм обнаружит все вершины, принадлежащие множеству E/I_1, связанные с выбранной tg-путем. Множество вершин, выбранное алгоритмом, обозначим через I_2. Это множество будет являться вторым мостом в графе.

Будем применять данную процедуру до тех пор, пока все вершины графа не будут распределены по своим островам. На некотором шаге N множество E/I_{N-1} станет пустым — это значит, что обнаружены все мосты в графе доступов.

4.2. Поиск мостов

После того как найдены все острова в графе доступов, необходимо выяснить, соединены ли они мостами. Согласно определению 4, мостом в графе доступов называется tg-путь, проходящий через вершины-объекты, концами которого являются вершины-субъекты; при этом словарная запись tg-пути имеет вид $\overrightarrow{t}^*$, $\overleftarrow{t}^*$, $\overrightarrow{t}^*\overrightarrow{g}\overleftarrow{t}^*$, $\overrightarrow{t}^*\overleftarrow{g}\overleftarrow{t}^*$.

Задачу отыскания моста в графе можно сформулировать следующим образом: необходимо построить алгоритм, который, анализируя вершины графа доступов и установленные права между ними, мог определить tg-путь заданного вида. Похожая задача решается для построения алгоритмов-распознавателей грамматик. Построение распознавателей грамматик подробно описывается в [4].

Построим распознаватель мостов в графе доступов. При этом мост $\overrightarrow{t}^*$ будем рассматривать как частный случай моста $\overrightarrow{t}^*\overrightarrow{g}\overleftarrow{t}^*$, то есть для поиска этих двух типов мостов будем использовать один алгоритм.

Введем ряд обозначений. Начальную и конечную вершину моста будем обозначать $S(e)$ и $F(e)$ соответственно. Вершину, в которой мы находимся в данный момент, будем обозначать $C(e)$, а рассматриваемую вершину — $W(e)$. Дугу между вершинами e_1 и e_2, содержащее право $\overrightarrow{g}$, обозначим $\overrightarrow{g}(e_1, e_2)$. Для дуг с правами $\overrightarrow{t}$ и $\overleftarrow{t}$ также введем соответствующие обозначения $\overrightarrow{t}(e_1, e_2)$ и $\overleftarrow{t}(e_1, e_2)$.

Для начала рассмотрим мосты $\overrightarrow{t}^*$ и $\overrightarrow{t}^*\overrightarrow{g}\overleftarrow{t}^*$.

Будем искать мост между островами графа I_1 и I_2. Пусть наш распознаватель находится в начальном состоянии, для которого определены вершины $S(e)$ и $F(e)$ ($S(e) \in I_1$, $F(e) \in I_2$). В самом начале работы $C(e) = S(e)$. Рассмотрим возможные варианты функционирования распознавателя. Очевидно,

что прямой дуги между $C(e)$ и $F(e)$, содержащей право t или g, быть не может, так как в этом случае обе вершины принадлежали бы одному острову. Для поиска моста между данными вершинами нам необходимо рассмотреть вершины-объекты графа доступов. Обозначим множество всех объектов через O. Выберем из этого множества вершину $W(e)$ и рассмотрим, каким образом эта вершина может быть связана с $C(e)$. Так как мы ищем мост $\overrightarrow{t}^*\overrightarrow{g}\overleftarrow{t}^*$, то нас интересует, существует ли дуга $\overrightarrow{t}$ или $\overrightarrow{g}$ между $C(e)$ и $W(e)$. Если такой дуги нет, то нам необходимо выбрать для рассмотрения следующую вершину $W(e)$ из O. Возможна ситуация, когда мы рассмотрели все вершины из O, но подходящей так и не нашли, в этом случае нам следует выбрать другую вершину $S(e) \in I_1$. Если же мы рассмотрели все вершины из I_1 и O, но так и не нашли ни одной подходящей нам пары, то, очевидно, моста не существует и распознаватель переходит в конечное состояние с отрицательным результатом. Назовем это состояние E^-. Допустим, что, перебирая пары $C(e)$ - $W(e)$, мы обнаружили дугу содержащую право $\overrightarrow{t}$ ($\overrightarrow{t}(C(e), W(e))$, это значит, что мы нашли дугу, которая может являться частью моста. Нам необходимо запомнить эту дугу, для этого пометим ее и вершину $W(e)$ *положительно*, сделаем рассматриваемую вершину текущей ($W(e) = C(e)$) и переведем распознаватель в новое состояние, которое будем называть $\overrightarrow{T}^+$. Нами осталась не рассмотрена ситуация, когда, перебирая пары $C(e)$ - $W(e)$, мы обнаружили дугу содержащую право $\overrightarrow{g}$ ($\overrightarrow{g}(C(e), W(e))$. В этом случае пометим дугу и вершину $W(e)$ положительно, выполним присвоение $W(e) = C(e)$ и переведем распознаватель в новое состояние, которое будем называть $\overrightarrow{G}^+$.

Допустим, что в ходе своего функционирования распознаватель перешел в состояние $\overrightarrow{T}^+$. В первую очередь нам необходимо проверить, существуют ли tg-дуги между текущей и конечной вершиной моста. Пусть существует дуга $\overrightarrow{t}(C(e), F(e))$, это означает, что распознаватель обнаружил мост типа $\overrightarrow{t}^*$; необходимо пометить эту дугу и перевести распознаватель в новое состояние, которое назовем E^+. Другой вариант предполагает, что существует дуга $\overrightarrow{g}(C(e), F(e))$, это означает, что распознаватель обнаружил мост типа $\overrightarrow{t}^*\overrightarrow{g}$. Пометим данную дугу и вершину положительно и переведем распознаватель в состояние E^+. Рассмотрим варианты, когда tg-дуги между текущей и конечной вершиной моста не существует. В этом случае распознаватель выбирает вершину $W(e) \in O$ из тех, что еще не помечены. Пусть существует дуга $\overrightarrow{t}(C(e), W(e))$ — пометим дугу и вершину $W(e)$ положительно, сделаем рассматриваемую вершину текущей ($C(e) = W(e)$); распознаватель остается в состоянии $\overrightarrow{T}^+$ и выбирает следующую вершину $W(e) \in O$. Если существует дуга $\overrightarrow{g}(C(e), W(e))$ — пометим дугу и вершину положительно, сделаем рассматриваемую вершину текущей и переведем распознаватель в состояние $\overrightarrow{G}^+$. Наконец, возможна ситуация, когда мы перебрали все непомеченные вершины из множества O и не нашли дуг $\overrightarrow{t}$ или $\overrightarrow{g}$ — это значит, что мы зашли в тупик. Попробуем вернуться к предыдущей вершине и выбрать другую, отличную от текущей. Для этого переведем распознаватель в состояние, которое будем называть $\overrightarrow{T}^-$.

Рассмотрим состояние $\overrightarrow{T}^-$. Распознаватель попадает в него из состояния $\overrightarrow{T}^+$

в том случае, если, рассмотрев все непомеченные вершины множества O, он не нашел дуг $\overrightarrow{t}$ или $\overrightarrow{g}$ между вершиной $C(e)$ и $W(e)$. Попробуем выбрать вершину, отличную от текущей. Для этого пометим $C(e)$ *отрицательно*, сделаем текущей предыдущую помеченную положительно вершину ($C(e) = C(e-1)$) и вернем распознаватель в состояние $\overrightarrow{T}^+$. Текущую вершину необходимо пометить отрицательно для того, чтобы распознаватель вновь не выбрал ее в состоянии $\overrightarrow{T}^+$. Возможна ситуация, когда, выбирая предыдущую вершину, мы попадем в вершину $S(e)$. Это, очевидно, означает, что между вершинами $S(e)$ и $F(e)$ не существует моста и необходимо вернуть распознаватель в начальное состояние для выбора других $S(e)$ и $F(e)$.

Пусть в ходе своей работы распознаватель перешел в состояние $\overrightarrow{G}^+$. Проверим, существует ли дуга $\overleftarrow{t}$ между вершинами $C(e)$ и $F(e)$. Если да, то это означает, что обнаружен мост $\overrightarrow{t}^*\overrightarrow{g}\overleftarrow{t}^*$. Пометим данную дугу положительно и переведем распознаватель в состояние E^+. Если нужной нам дуги между $C(e)$ и $F(e)$ не существует, то необходимо выбрать непомеченную вершину $W(e)$ из множества O. Если существует дуга $\overleftarrow{t}\,(C(e), W(e))$, то пометим эту дугу и вершину W(e) положительно, присвоим $C(e) = W(e)$, и переведем распознаватель в новое состояние, которое будем называть $\overleftarrow{T}^+$. Все прочие варианты нас не интересуют, однако возможна такая ситуация, когда мы перебрали все непомеченные вершины из множества O, но так и не нашли дуги, содержащей право $\overleftarrow{t}$. В этом случае нам придется пометить текущую вершину отрицательно, выбрать в качестве текущей предыдущую помеченную положительно ($C(e) = C(e-1)$) и вернуться в состояние $\overrightarrow{T}^+$.

Если распознаватель попал в состояние $\overleftarrow{T}^+$, то нам следует проверить, существует ли дуга $\overleftarrow{t}\,(C(e), F(e))$. Если это так, то это значит, что обнаружен мост $\overrightarrow{t}^*\overrightarrow{g}\overleftarrow{t}^*$; пометим данное ребро положительно и перейдем в состояние E^+. В противном случае выберем непомеченную вершину $W(e) \in O$ и проверим, существует ли дуга $\overleftarrow{t}\,(C(e), W(e))$. Если такая дуга существует, то пометим вершину $W(e)$ и дугу положительно, присвоим $C(e) = W(e)$ и вновь перейдем в состояние $\overleftarrow{T}^+$. В случае, когда, перебрав все непомеченные вершины из O, мы не нашли нужной дуги (попали в тупик), необходимо попробовать вернуться к предыдущей помеченной положительно вершине. Для этого переведем распознаватель в новое состояние, которое назовем $\overleftarrow{T}^-$.

Опишем состояние $\overleftarrow{T}^-$. Распознаватель переходит сюда из состояния $\overleftarrow{T}^+$, если, перебрав все непомеченные вершины из O, он не смог обнаружить дуги $\overleftarrow{t}$ между $C(e)$ и $W(e)$. Пометим текущую вершину отрицательно, вернемся к предыдущей помеченной положительно вершине ($C(e) = C(e-1)$) и вновь перейдем в состояние $\overleftarrow{T}^+$. Если при выборе предыдущей вершины мы попадем в вершину, которой закончилась дуга $\overrightarrow{g}$, то нам придется пометить ее отрицательно и вернуться в состояние $\overrightarrow{T}^+$.

Графическая схема работы распознавателя представлена на рисунке 1.

Для поиска моста $\overrightarrow{t}^*\overleftarrow{g}\overleftarrow{t}$ можно воспользоваться описанным распознавателем, если заменить состояние $\overrightarrow{G}^+$ на состояние $\overleftarrow{G}^+$.

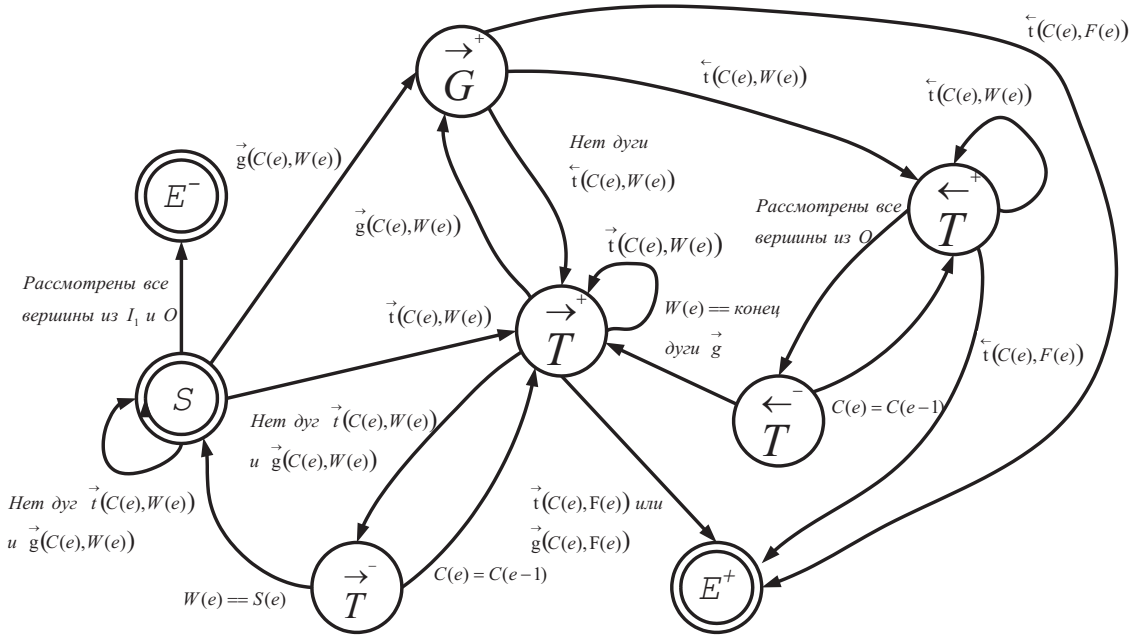

Рис. 1. Распознаватель моста $\vec{t}^{\,*}\ \vec{g}\ \overleftarrow{t}^{\,*}$

Построим распознаватель моста $\overleftarrow{t}^{\,*}$. Будем искать мост между островами I_1 и I_2. В начальном состоянии выберем $S(e) \in I_1$, $F(e) \in I_2$, присвоим $C(e) = S(e)$. Очевидно, что в данном случае не существует дуги $\overleftarrow{t}\,(C(e), F(e))$, иначе вершины $S(e)$ и $F(e)$ принадлежали бы одному острову. Выберем вершину $W(e)$ из множества O и проверим, существует ли дуга $\overleftarrow{t}\,(C(e), W(e))$. Если дуга существует, то пометим данную дугу и вершину $W(e)$ положительно, присвоим $C(e) = W(e)$ и переведем распознаватель в состояние $\overleftarrow{T}^{+}$; иначе выберем следующую вершину $W(e)$. Если мы рассмотрели все вершины из O, но при этом так и не нашли ни одной дуги $\overleftarrow{t}\,(C(e), W(e))$, то попробуем выбрать другую вершину $S(e) \in I_1$. Наконец, когда мы перебрали все вершины из I_1 и не нашли ни одной дуги $\overleftarrow{t}\,(C(e), W(e))$ — это значит, что моста не существует; переведем распознаватель в состояние E^{-}.

Пусть распознаватель попал в состояние $\overleftarrow{T}^{+}$. Проверим, существует ли дуга $\overleftarrow{t}\,(C(e), F(e))$. Если да, то это значит, что обнаружен мост $\overleftarrow{t}^{\,*}$, пометим дугу положительно и переведем распознаватель в состояние E^{+}. Если такой дуги нет, то выберем вершину $W(e) \in O$ и проверим, существует ли дуга $\overleftarrow{t}\,(C(e), W(e))$. Если такая дуга есть, то пометим ее и вершину $W(e)$ положительно, присвоим $C(e) = W(e)$ и вновь переведем распознаватель в состояние $\overleftarrow{T}^{+}$; иначе выберем следующую вершину $W(e)$. Когда мы рассмотрим все вершины из O, но при этом не обнаружим ни одной дуги $\overleftarrow{t}\,(C(e), W(e))$, то попробуем вернуться к предыдущей помеченной положительно вершине, для этого переведем распознаватель в состояние $\overleftarrow{T}^{-}$.

В состоянии $\overleftarrow{T}^{-}$ необходимо пометить текущую вершину отрицательно и вернуться к предыдущей положительной вершине ($C(e) = C(e-1)$), после чего

нужно перевести распознаватель обратно в $\overleftarrow{T}^+$. Если при переходе к предыдущей вершине мы оказались в вершине $S(e)$, то необходимо вернуться в начальное состояние для выбора другой $S(e) \in I_1$.

Графическая схема работы распознавателя для поиска моста $\overleftarrow{t}^*$ представлена на рисунке 2.

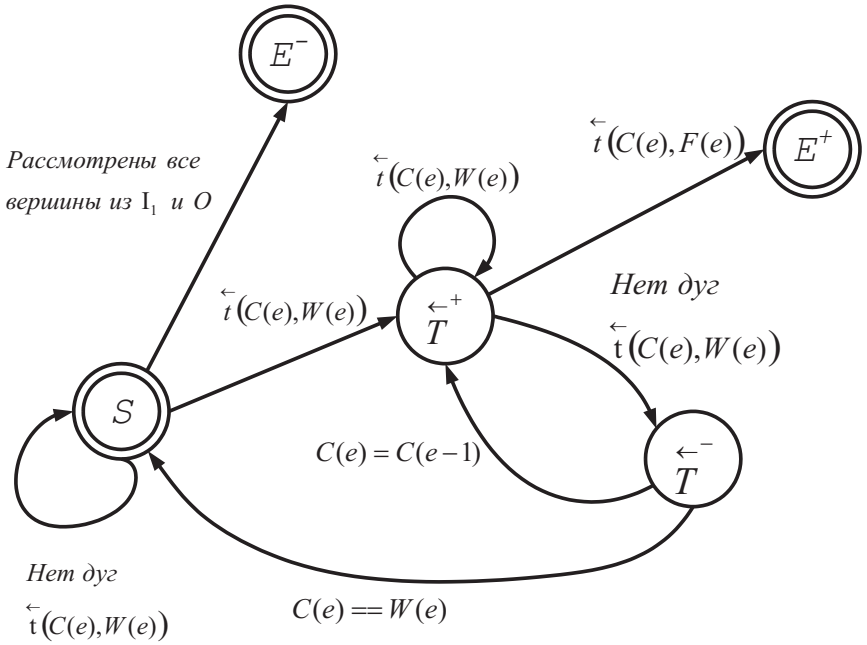

Рис. 2. Распознаватель моста $\overleftarrow{t}^*$

4.3. Поиск начального и конечного пролета моста

Нами остались не рассмотрены алгоритмы поиска начального и конечного пролета моста.

Рассмотрим поиск начального пролета. По определению 5 начальный пролет моста — это tg-путь, проходящий через вершины-объекты, началом которого является вершина-субъект, а концом — вершина-объект; при этом словарная запись пути имеет вид $\overrightarrow{t}^*\overrightarrow{g}$. Построим распознаватель начального пролета моста.

В первую очередь нам необходимо определить вершину-объект, которая будет являться концом пролета. Обозначим эту вершину $F(e)$. Будем искать начальный пролет моста от острова I до вершины $F(e)$. В начальном состоянии нам необходимо выбрать вершину $S(e) \in I$. Также нужно выбрать рассматриваемую вершину, в начальном состоянии это будет вершина $S(e)$ $(C(e) = S(e))$. После этого проверим, существует ли дуга $\overrightarrow{g}(C(e), F(e))$. Если такая дуга существует — это значит, что начальный пролет моста обнаружен, пометим дугу положительно и переведем распознаватель в состояние E^+. Если такой дуги нет, то выберем вершину $W(e) \in O$ и проверим, существует ли дуга $\overrightarrow{t}(C(e), W(e))$.

Если дуга есть, то пометим ее и вершину $W(e)$ положительно, выполним присвоение $C(e) = W(e)$ и перейдем в состояние $\overrightarrow{T}^+$. Если дуги нет, то выберем следующую вершину $W(e)$. Если при переборе всех вершин из O дуги $\overrightarrow{t}(C(e), W(e))$ не обнаружится, то выберем другую вершину $S(e)$. Когда будут рассмотрены все вершины из I, но при этом распознаватель будет находиться в начальном состоянии — мы выясним, что начального пролета между островом I и вершиной $F(e)$ не существует; переведем распознаватель в состояние E^-.

Если распознаватель находится в состоянии $\overrightarrow{T}^+$, то в первую очередь нужно проверить, существует ли дуга $\overrightarrow{g}(C(e), F(e))$. Если существует, то, очевидно, мы нашли начальный пролет моста; пометим дугу положительно и перейдем в состояние E^+. Если такой дуги нет, то выберем следующую непомеченную вершину $W(e)$ из O, проверим, существует ли дуга $\overrightarrow{t}(C(e), W(e))$. Так же, как и в предыдущих рассмотренных случаях, если такая дуга есть, то помечаем ее и рассматриваемую вершину положительно, выполняем присвоение $C(e) = W(e)$ и вновь переходим в состояние $\overrightarrow{T}^+$. Если же дуги нет, то переходим в состояние $\overrightarrow{T}^-$.

Состояние $\overrightarrow{T}^-$ также аналогично ранее рассмотренным. Здесь мы помечаем текущую вершину отрицательно, переходим к предыдущей положительной вершине ($C(e) = C(e-1)$) и возвращаемся в состояние $\overrightarrow{T}^+$. Если $C(e-1) == S(e)$, то переходим в начальное состояние.

Граф работы распознавателя представлен на рисунке 3.

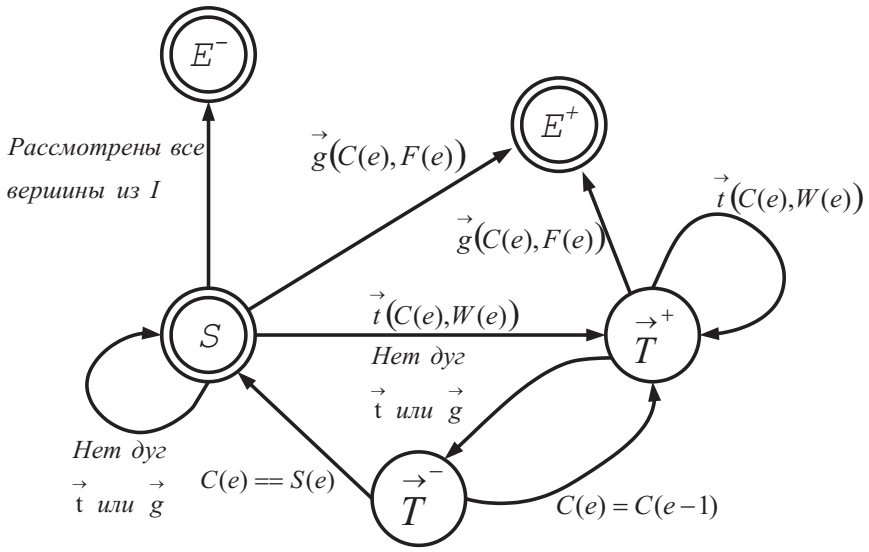

Рис. 3. Распознаватель начального пролета моста

Наконец, рассмотрим алгоритм для поиска конечного пролета моста. По определению 6, конечный пролет моста — это tg-путь, проходящий через вершины-объекты, началом которого является вершина-субъект, а концом — вершина-объект; при этом словарная запись пути имеет вид $\overrightarrow{t}^*$.

Как и в предыдущих случаях, построим распознаватель для данного типа tg-пути. Для начала выберем конечную вершину моста из $F(e) \in O$ и началь-

ную вершину $S(e) \in I$. В начальном состоянии присвоим $C(e) = S(e)$ проверим наличие дуги $\overrightarrow{t}(C(e), F(e))$. Если дуга есть, то пометим ее положительно и перейдем в состояние E^+ (конечный пролет найден), если нет — выберем вершину $W(e) \in O$. Проверим наличие дуги $\overrightarrow{t}(C(e), W(e))$, если такая дуга есть — перейдем в состояние $\overrightarrow{T}^+$, в противном случае выберем следующую $W(e)$. Вновь стоит учесть ситуацию, когда при переборе всех вершин из O мы не нашли нужной дуги, в этом случае нужно выбрать следующую вершину $S(e) \in I$. Не забудем также, что в случае перебора всех субъектов из I и всех объектов из O мы можем не найти ни одной дуги $\overrightarrow{t}(C(e), W(e))$ и тогда, следует перевести распознаватель в состояние E^-.

Состояние $\overrightarrow{T}^+$ аналогично ранее рассмотренным. Здесь проверяется наличие дуги $\overrightarrow{t}(C(e), F(e))$, и в положительном случае распознаватель переводится в состояние E^+. В отрицательном случае проверяется наличие дуги $\overrightarrow{t}(C(e), W(e))$, и если дуга есть, то она и $W(e)$ помечаются положительно, рассматриваемая вершина делается текущей ($C(e) = W(e)$), и распознаватель возвращается в $\overrightarrow{T}^+$. Если перебрали все вершины из O, а дуги $\overrightarrow{t}(C(e), W(e))$ не нашли, то переходим в состояние $\overrightarrow{T}^-$.

В состоянии $\overrightarrow{T}^-$ переходим к предыдущей помеченной положительно вершине и возвращаемся в $\overrightarrow{T}^+$. Также не следует забывать про возможность перехода в начальное состояние в случае, когда $C(e-1) == S(e)$.

Граф работы распознавателя представлен на рисунке 4.

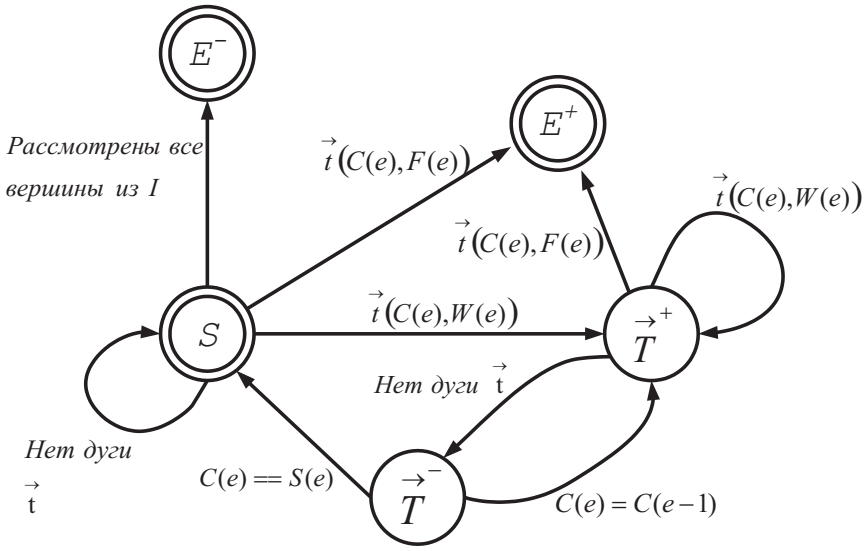

Рис. 4. Распознаватель конечного пролета моста

Мы рассмотрели алгоритмы для поиска tg-путей, островов и мостов в графе доступов. Теперь мы можем построить формальный автомат, который смог бы выяснить, является ли предикат *«возможен доступ»*(α, x, y, G_0) истинным для данного состояния системы. То есть этот автомат может показать, является ли данное состояние компьютерной системы безопасным.

Литература

1. Девянин, П.Н. Модели безопасности компьютерных систем: Учебное пособие для студентов высших учебных заведений / П.Н. Девянин. – М.: Издательский центр «Академия», 2005.
2. Гайдамакин, Н.А. Разграничение доступа к информации в компьютерных системах / Н.А. Гайдамакин. – Е.: Издательство Уральского университета, 2003.
3. Майника, Э. Алгоритмы оптимизации на сетях и графах / Э. Майника – М.: Издательство «Мир», 1981.
4. Гордеев, А.В. Системное программное обеспечение. Учебник / А.В. Гордеев, А.Ю. Молчанов – СПб.: «Питер», 2001.

Математические структуры и моделирование
2009, вып. 20, с. 173–177

УДК 519.71:371.3

КУРС «ОСНОВЫ ТЕОРИИ УПРАВЛЕНИЯ» ДЛЯ ИНЖЕНЕРОВ-ПРОГРАММИСТОВ

И.П. Бесценный, А.А. Лаптев

Обсуждаются проблемы преподавания курса «Основы теории управления» на факультете компьютерных наук и предлагаются методы их решения, исходя из реального опыта преподавателей факультета.

Согласно государственному образовательному стандарту для специальности 230101 «Вычислительные машины, комплексы, системы и сети» дисциплина «Основы теории управления»(далее ОТУ) отнесена к блоку общепрофессиональных дисциплин и принадлежит федеральной компоненте образовательной программы. Этот статус говорит о достаточно большой степени фундаментализации при разработке рабочей программы ОТУ [3]. Из принципов фундаментализации образования непосредственно от преподавателей вуза зависит модульность программ учебных дисциплин, развитие системы междисциплинарных спецкурсов, сбалансированность аудиторного и самостоятельного обучения, сближение между высшей и средней школой.

Если подразумевать теорию управления в широком смысле этого слова, то следует включить в содержание дисциплины общий обзор современных достижений всей кибернетики (как известно, кибернетика и есть наука об управлении). Однако, анализ содержания рабочих программ ОТУ в других вузах [2, 9–11, 16, 18] показывает, что подавляющее большинство времени отводится на изучение только теории линейных систем автоматического управления (с примерами в основном из радиоэлектроники), которая является лишь узким разделом технической (а не всей) кибернетики.

Конечно, в узкой предметной области можно найти больше существенных закономерностей, чем когда объектом исследования являются сложные системы реального мира, где сложность определяется принципиальной невозможностью детального описания [7]. Тем не менее инженер по специальности 230101 «Вычислительные машины, комплексы, системы и сети» должен знать и уметь применять различные модели и методы анализа и проектирования сложных информационных систем, как следует из требований к уровню подготовки дипломированного специалиста. Помимо систем автоматического управления, в курс ОТУ следует включать оптимальное управление, исследование недетерминированных систем, самонастраивающиеся системы управления, алгоритмы

Copyright © 2009 **И.П. Бесценный, А.А. Лаптев.**
Омский государственный университет.

цифрового управления. За редким исключением ([2,10,13]) в учебниках и учебных пособиях по теории управления этим вопросам уделяется мало внимания. С другой стороны, отметим, что в учебной литературе по теории систем и системному анализу для других специальностей уделяется место и принципам и методам теории управления, вплоть до описания типовых линейных звеньев [8]. Так как дисциплина «Теория систем» федеральной компонентой образовательного стандарта по специальности 230101 не предусмотрена, то та часть, которая является общей по смыслу для теории систем и теории управления, должна или входить в курс ОТУ, или преподаваться в качестве региональной (вузовской) компоненты. Программа такого спецкурса сейчас разрабатывается на нашем факультете.

Итак, первая проблема при разработке программы ОТУ – тщательный отбор материала, который следует знать студентам.

Достаточно полный комплект учебников по различным разделам теории управления [1, 4, 5, 12, 14, 15] содержит несколько тысяч страниц, и его подробное изложение невозможно не то что в семестровом, даже в годовом курсе. Ограниченность общей трудоёмкости дисциплины (120 часов), стремление к фундаментализации и целостности курса приводят к тому, что изложение материала охватывает широкий круг вопросов, но недостаточно глубоко. Такая поверхностность изложения вместе с недостаточной подготовленностью студентов приводит к поверхностному усвоению материала, что отражается на экзаменационных оценках. Математический аппарат, используемый в теории управления, тоже обширен [6]. Для освоения курса ОТУ студент должен знать следующие разделы математики:

- операции с комплексными числами и функциями;
- алгебра матриц;
- разложение функции в ряд Тейлора и Лорана;
- разложение функции в ряд Фурье;
- решение дифференциальных уравнений с помощью преобразования Лапласа;
- случайные процессы;
- нахождение экстремума функционала.

Если студенты очной формы обучения проходят ОТУ на третьем курсе, после 2,5 лет учебы на факультете, то студенты сокращенной формы обучения получают базовое образование в техникумах. Уровень их подготовки сильно разнится. Поэтому на лекциях приходится планировать время на дополнительное пояснение материала.

Вторая проблема при преподавании ОТУ – согласование рабочих программ соответствующих математических дисциплин.

Когда материал отобран и согласован по объёму с другими дисциплинами, остаётся его расположить в нужном порядке. Есть два способа изложения – от общего к частному и наоборот. Математикам свойственно первое, инженерам – второе. Хотя специальность «Вычислительные машины, комплексы, системы и сети» является инженерной, на факультете компьютерных наук она имеет более математическо-программистский уклон, чем инженерно-электротехнический. Поэтому акцент делается на изложение от общего к частному. Но в отдельных случаях методически полезным оказывается сначала рассмотреть примеры физически различных систем управления (механических, гидравлических, электрических), а затем вывести общие для них принципы и единую математическую модель. На практических занятиях постоянно используется программа симуляции систем управления VisSim (рис. 1). Мы считаем полезным сравнение чисто математических расчётов поведения системы управления с наглядным графическим представлением её симуляции. Программа VisSim удобна в использовании, компактна и легко осваивается студентами самостоятельно. Она позволяет охватить примерами практически все разделы, рассматриваемые в курсе ОТУ. Есть у пакета VisSim и недостатки: невозможность сохранения полученных графических результатов, не всегда удовлетворительная точность численных вычислений и др.

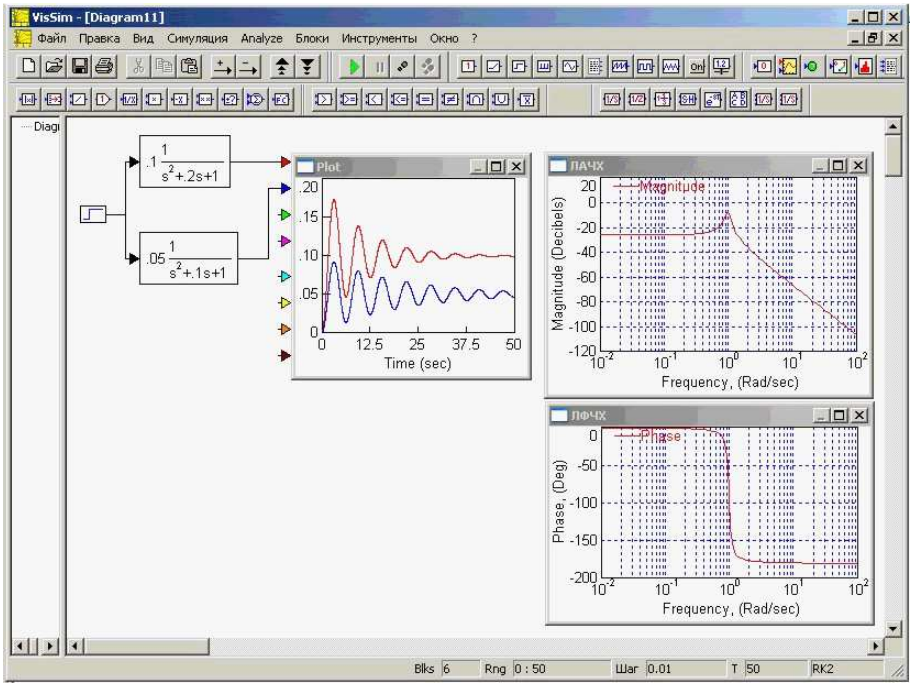

Рис. 1. Окно программы VisSim

Таким образом, третья проблема при построении курса ОТУ – адекватный подбор примеров для лекционного материала и для практических заданий, эффективное сочетание теоретического материала и практических заданий.

Все эти проблемы решаются при разработке рабочих программ курса «Основы теории управления» и при подготовке к занятиям преподавателями фа-

культета компьютерных наук. С точки зрения подачи лекционного материала курс ОТУ достаточно сложен, так как требуется воспроизведение большого графического материала (схемы, графики и т.п.) для наглядности излагаемого материала. С 2009 года преподавание ведется с помощью презентаций, отображаемых на экране в аудитории. Это позволило упростить процесс изложения и повысить качество подачи материала. Использование проектора дает возможность проводить показ работы моделей в реальном режиме.

На факультете компьютерных наук почти половина студентов обучается на сокращенной форме обучения. Количество аудиторных часов сокращено на 30% у очной формы обучения, а на заочной форме – на 70%. Поэтому большая часть материала должна осваиваться студентами самостоятельно. Для заочной формы обучения набор лабораторных работ сокращен и упрощена форма представления результатов. Студенты-заочники на зимней сессии второго курса прослушивают установочные лекции и получают задания для лабораторных работ, часть которых они успевают выполнить на той же сессии. В летнюю сессию проходит сдача лабораторных работ, которые они выполняют самостоятельно. Во время приема работ проверятся правильность их выполнения, понимание студентом изученного материала. Студенты-заочники имеют некоторое преимущество в том, что есть четкие требования для получения положительной оценки. К сожалению, большинство студентов сокращенной формы обучения ограничиваются выполнением минимальных требований для получения оценки «удовлетворительно», и только 20-30% студентов делают усилия для получения более высокой оценки.

Студенты факультета имеют большие возможности для самостоятельной работы: лабораторные работы выдаются в электронном виде, в распоряжение студентов предоставлено пять компьютерных классов, в которых выделено время для индивидуальной работы. При этом задания для лабораторных работ постоянно изменяются и совершенствуются с целью исключения копирования результатов недобросовестными студентами.

Таким образом, курс ОТУ адаптируется под специфику нашего факультета для повышения качества образования.

Литература

1. Астапов, Ю.М. Статистическая теория систем автоматического регулирования и управления / Ю.М. Астапов, В.С. Медведев. – М.: Наука, 1982.
2. Афанасьев, В.А. Теория автоматического управления / В.А. Афанасьев и др. / Уч.пособие: в 2 ч. – Ижевск: ИжГТУ, 2007.
3. Бесценный, И.П. О проблемах фундаментального естественно-научного образования / И.П. Бесценный // Естественные науки в ОмГАУ. – Омск: ОмГАУ, 2005. – С. 142-152,
4. Иванов, В.А. Теория оптимальных систем автоматического управления / В.А. Иванов, Н.В. Фалдин. – М.: Наука, 1981.
5. Иванов, В.А. Теория дискретных систем автоматического управления / В.А. Иванов, А.С. Ющенко. – М.: Наука, 1983.

6. Иванов, В.А. Математические основы теории автоматического регулирования / В.А. Иванов и др. – М.: Высшая школа, 1971.
7. Калман, Р. Очерки по математической теории систем / Р. Калман и др. – М.: Мир, 1971.
8. Качала, В.В. Основы теории систем и системного анализа / В.В. Качала. – М.: Горячая линия Телеком, 2007.
9. Клиначев, Н.В. Теория систем автоматического регулирования / Н.В. Клиначев [Электронный ресурс]. – Режим доступа: http://model.exponenta.ru/lectures (2004).
10. Леонов, Г.А. Введение в теорию управления / Г.А. Леонов. – СПб.: СпбГУ, 2004.
11. Никулин, Е.А. Основы теории автоматического управления / Е.А. Никулин. – СПб.: БХВ-Петербург, 2000.
12. Оппенгейм, Э. Цифровая обработка сигналов / Э. Оппенгейм, Р. Шафер / Пер. с англ. – М.: Техносфера, 2007.
13. Пантелеев, А.В. Теория управления в примерах и задачах / А.В. Пантелеев, А.С. Бортаковский. – М.: 2003.
14. Попов, Е.П. Теория линейных систем автоматического регулирования и управления / Е.П. Попов. – М.: Наука, 1989.
15. Попов, Е.П. Теория нелинейных систем автоматического регулирования и управления / Е.П. Попов. – М.: Наука, 1988.
16. Фалеев, М.В. Теория автоматического управления / М.В. Фалеев [Электронный ресурс]. – Режим доступа: http://elib.ispu.ru/library/lessons/faleev (2004).
17. Федосов, Б.Т. Задания и методические указания к выполнению лабораторных работ по курсам «Системотехника» и «ТАУ» / Б.Т. Федосов [Электронный ресурс]. – Режим доступа: http://model.exponenta.ru/bt (2005).
18. Юревич, Е.И. Теория автоматического управления / Е.И. Юревич. – СПб.: БХВ-Петербург, 2007.

Математические структуры и моделирование
2009, вып. 20, с. 178–190

УДК 004.045+372.862

ОТ ИМПЕРАТИВНОГО К ОБЪЕКТНО-ОРИЕНТИРОВАННОМУ ПРОГРАММИРОВАНИЮ ВМЕСТЕ С JAVA И NETBEANS: ОБЪЕКТНАЯ ДЕКОМПОЗИЦИЯ И ИНКАПСУЛЯЦИЯ

Д.Н. Лавров

Основной проблемой при переходе от императивного к объектно-ориентированному программированию является изменение стиля мышления. Как изменить этот стиль за одно-два занятия и при этом продемонстрировать основные подходы, используемые при разработке, возникающие сложности и способы их решения — это цель данной статьи. На примере решения простейшей задачи мы ведем учащегося по дороге от умения составлять алгоритмы к способностям строить масштабируемую архитектуру приложения.

Введение

Объектно-ориентированное программирование (ООП) — это один из способов бороться с увеличивающейся сложностью программного обеспечения. Реальные ощутимые плоды этот подход дает на больших проектах, а на небольших задачах применение подхода приводит к увеличению объема кода. Учащиеся, видя увеличения объема, часто теряют мотивацию в использовании ООП. Эту мотивацию нужно создавать, объясняя, как важно в больших программных проектах правильно провести объектную декомпозицию.

К сожалению, приводить примеры больших проектов на лабораторных работах и выдавать задания больших объемов на самостоятельную разработку невозможно. Из опыта преподавания известно, что такие проекты, как правило, не завершаются удачно.

Учиться нужно на небольших примерах. Разобрав пример, необходимо дать похожее по уровню сложности задание учащемуся и проконтролировать его выполнение.

Основной критерий — правильная объектная декомпозиция задачи. К сожалению или к счастью, не существует единственно верного решения. Каждое из

Copyright © 2009 **Д.Н. Лавров.**
Омский государственный университет.
E-mail: lavrov@omsu.ru

решений будет иметь свои достоинства и недостатки. Существуют откровенно неверные решения и решения, верные при рассмотрении с каких-либо позиций.

Пусть необходимо создать приложение, вычисляющее действительные корни квадратного уравнения с действительными коэффициентами.

Из каких соображений будем исходить при решении этой задачи:

1. Необходимо скрыть реализацию классов друг от друга и от программиста, пользователя класса. Другими словами, обеспечить инкапсуляцию.

2. Необходимо построить приложение, в котором логика выполняемой работы не смешивалась бы с интерфейсной частью. Здесь можно также говорить об инкапсуляции, но уже не на уровне класса, а на уровне всего приложения. Скрывается реализация логики работы приложения от интерфейсных классов.

1. Решение в императивном стиле

С чего начать? Попросим студентов создать программу на языке Java, которая решает задачу нахождения корней квадратного уравнения. Стиль интерфейса не важен, пусть это будет консольный ввод-вывод. Скорее всего получиться код, подобный следующему:

```
import static java.lang.Math.*;
import java.io.*;
public class SolverSquareEquation {
    public static void main(String[] args) throws IOException {
        int numberOfRoots;
        double a, b, c, x1=0, x2=0;
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.print("a=");
        a=Double.parseDouble(br.readLine());
        System.out.print("b=");
        b=Double.parseDouble(br.readLine());
        System.out.print("c=");
        c=Double.parseDouble(br.readLine());
        if (a!=0) {
            double d=b*b-4*a*c;
            if        (d<0) {numberOfRoots=0;}
            else if  (d==0) {numberOfRoots=1;  x1=-b/(2*a); }
            else /*(d>0)*/  {numberOfRoots=2;  x1=(-b-sqrt(d))/(2*a);
                                               x2=(-b+sqrt(d))/(2*a);}
        } else /*(a==0)*/ {
            if      (b==0&&c==0)  {numberOfRoots=-1;}
            else if (b==0&&c!=0)  {numberOfRoots= 0;}
            else  /*(b!=0&&c!=0)*/{numberOfRoots= 1; x1=-c/b;}
        }
        switch (numberOfRoots){
```

```
            case -1: {System.out.println("X-любое число.");break;}
            case  0: {System.out.println("Корней нет.");break;}
            case  1: {System.out.println("X1=");break;}
            case  2: {System.out.println("X1="+x1+" X2="+x2);break;}
        }
    }
}
```

В этом примере разобраны все случаи входных данных. Конечно, не разобравший всех случаев учащийся должен исправиться, ведь программа должна работать на всех типах входных данных – и правильных, и неправильных.

Обратите внимание на то, что в данной программе сложное действие не разбито на более простые, и можно сказать, что программа на объектно-ориентированном языке скорее написана даже не в императивном, а в структурном стиле. Плохо ли это? Смотря для чего. Если цель просто решить квадратное уравнение, то нет. Но наша цель — освоить новый способ мышления, поэтому не будем останавливаться на достигнутом.

Что характеризует полученное решение:

- Это проектное решение невозможно тестировать, кроме как в «ручном» режиме, путем последовательного ввода данных и визуальной проверке полученных ответов.
- Это решение трудно поддерживать, оно не поддается стандартным методам рефакторинга [2].
- Это решение фактически не использует объектов, а значит, невозможно получить преимущества повторного использования и расширяемости без переписывания кода.
- Ни о каком разделении интерфейсной части кода и кода, представляющего решение основной задачи, речи не идет: код взаимодействия с консолью и код решения основной задачи приложения сплетены между собою, а следовательно, в будущем возникнут трудности при попытке смены интерфейса с консольного на оконный или web-интерфейс.

В таких случаях говорят, что приложение немасштабируемое.

Прежде чем исправить указанные недостатки, обсудим преимущества объектно-ориентированного подхода.

2. Суть объектно-ориентированного подхода

Основная идея заключается в построении модели на основе выделения понятий предметной области с последующим распределением обязанностей между ними. Программа, решающая нашу задачу, строится так, как задача решалась бы в жизни. По сути, строится модель предметной области, понятия предметной области представляются в виде классов, а их конкретные проявления в виде объектов. Распределяя обязанности между классами, проектировщик приложения

«оживляет» объекты, наделяет их поведением. Оживление (Animation) является основным принципом, используемым при построении архитектуры объектно-ориентированного приложения [1]. Проектировщик (программист) уподобляется богу, создающему искусственный мир по образу и подобию настоящего мира.

Моделирование осуществляется декомпозицией не по действиям (глаголам), как было принято в императивном программировании, а по сущностям (существительным). Сущности представляют собой, как правило, понятия предметной области, но иногда искусственные понятия, которых нет в явном виде в предметной области, но без которых невозможно создать масштабируемый код.

При оживлении пользователю объектов совсем не обязательно знать внутреннюю структуру объекта, так же как в реальной жизни пользователь обычно не знает, как в точности устроена микроволновая печь или телевизор, что не мешает ему разогревать еду и просматривать телепередачи. Скрытие внутренней реализации объекта от пользователя – это обычная практика, которая называется инкапсуляцией. Важна только функциональность объекта, его интерфейс. Такой подход позволяет ограничить распространение изменений в коде класса на все приложение. Это один из важнейших принципов объектно-ориентированного проектирования.

Необходимо сломать установку учащегося, которая скрывается за высказыванием: «*Но ведь программа работает правильно*». «Работает правильно» - это необходимое условие хорошей программы, но, к сожалению, недостаточное.

3. Объектная декомпозиция задачи

Класс представляет собой некую сущность, которая задает некоторое общее поведение для объектов. Разработка или, точнее, моделирование приложения состоит из двух этапов: анализа и проектирования. В большинстве современных процессов разработки результаты этих двух этапов описываются на графическом языке UML [3] в виде диаграмм.

Цель анализа — выявление объектов и описывающих их классов, представляющих важные, с точки зрения разработчика и заказчика, понятия предметной области. Обычно выделенные понятия формируют так называемую концептуальную модель, которая становится источником большинства классов и объектов разрабатываемого приложения [1].

Выделение объектов и понятий предметной области (будущих классов) часто основывается на выделении существительных из текстовых описаний работы будущего приложения. В нашем случае можно выделить три таких понятия (класса):

SquareEquation – квадратное уравнение;

Solution – решение квадратного уравнения;

SolverSquareEquation – тот кто, пользуется услугами решения квадратного уравнения. Это будет интерфейсный класс, который осуществляет взаимодействие с пользователем через консоль.

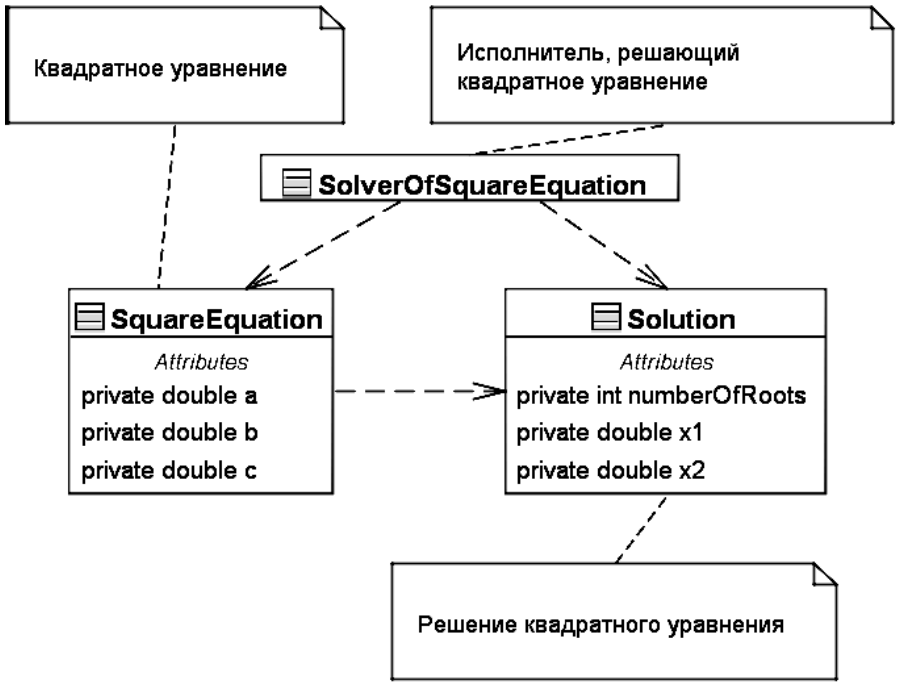

Рис. 1. Концептуальная модель задачи решения квадратного уравнения, представляется на диаграмме классов

Следующая задача этапа анализа — определение наиболее важных атрибутов и ассоциаций (связей) между понятиями. В нашем случае атрибуты достаточно очевидны, а постоянных ассоциаций нет.

Все полученные результаты сводятся на диаграмму классов, представляющую концептуальную модель приложения (рис. 1). На этой диаграмме пунктирные стрелки означают использование одного класса другим. Такие стрелки применяют, если необходимо указать, что экземпляр класса является локальной переменной, параметром или возвращаемым значением метода класса, от которого берет начало стрелка.

Цель проектирования — это распределение обязанностей между объектами для удовлетворения всех требований решаемой задачи. Результаты такого распределения представляются в виде UML-диаграмм взаимодействий, описывающих последовательность сообщений между объектами и UML-диаграммы классов, представляющих архитектуру будущего приложения. На этом этапе понятия предметной области превращаются в программные классы.

Итак, в нашей задаче на этапе проектирования необходимо решить, какому объекту (классу) поручить решать квадратное уравнение. Существует несколько вариантов решения этой задачи, но если необходимо сократить объем кода приложения, то нужно передать эту обязанность тому классу, который обладает наибольшим количеством информации для выполнения задачи. Такой способ решения поставленной задачи носит название шаблона распределения обязанностей Expert, относящегося к семейству шаблонов GRASP [1]. В нашем случае

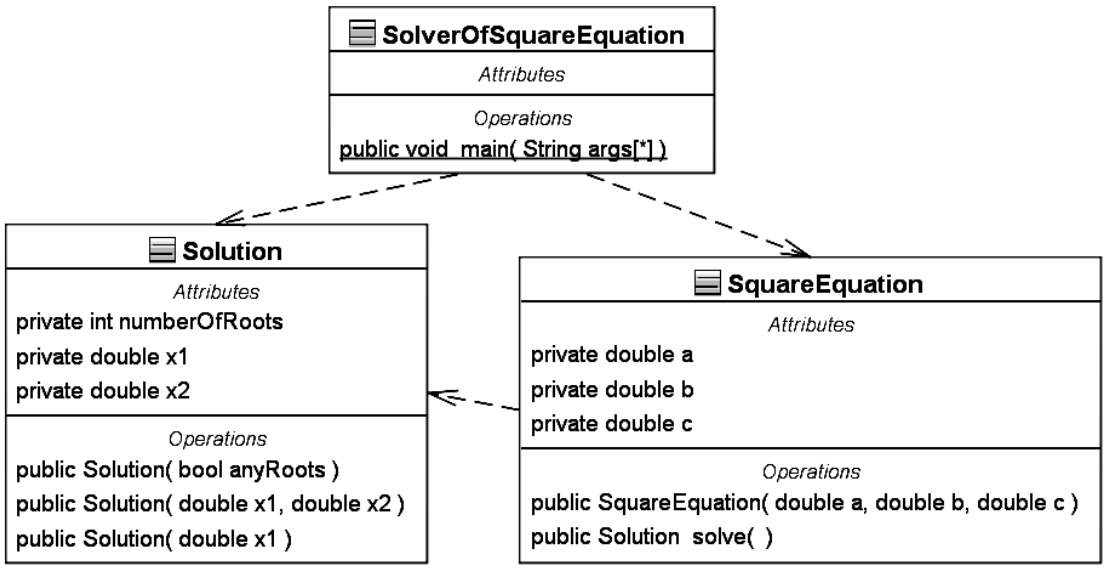

Рис. 2. UML-диаграмма классов, представляющая архитектуру приложения SolverSquareEquation

таким классом является класс SquareEquation, ведь именно он обладает всеми данными для получения решения квадратного уравнения. В результате метод solve() становиться методом SquareEquation.

Полезно разобрать с учащимися другие решения распределения обязанностей и посмотреть, к какому коду и каким проблемам могут эти решения привести.

Далее следует определить, какой класс будет отвечать за консольный интерфейс приложения. Наиболее подходящим для этих целей является класс SolverOfSquareEquation, представляющий все приложение в целом. Реализация ввода/вывода будет осуществляться в статическом методе main() так, что создавать экземпляр этого класса не нужно.

После распределения обязанностей получаем диаграмму классов (рис. 2).

Обратите внимание, что для решения нашей задачи нам не нужны все методы доступа и инициализации (методы типа get/set). Все лишнее должно быть отброшено, чтобы код был более ясным и не загромождался никому не нужными конструкциями.

Интегрированная среда разработки NetBeans[1] позволяет построить такую диаграмму классов, а затем сгенерировать по ней код, в котором останется только дописать реализации методов. Нормально, если в диаграмме классов будут опушены некоторые детали или будут присутствовать несущественные недоработки. В дальнейшем при реализации можно будет исправить эти недоработки в коде, а на следующей итерации разработки методом обратного реинженеринга (также реализован в NetBeans) скорректировать вышеупомянутую диаграмму классов.

[1] Свободно распространяемая среда разработки, доступная по адресу http://www.netbeans.org

После кодирования и внесения доработок получаем следующий код:

```
// модуль компиляции SquareEquation
import static java.lang.Math.*;
public class SquareEquation {
    private double a;
    private double b;
    private double c;

    public SquareEquation (double a, double b, double c) {
        this.a=a;
        this.b=b;
        this.c=c;
    }

    public Solution solve () {
       if (a!=0) {
            double d=b*b-4*a*c;
            if        (d<0) {return new Solution(false);}
            else if  (d==0) {return new Solution(-b/(2*a)); }
            else /*(d>0)*/  {return new Solution((-b-sqrt(d))/(2*a),
                                                 (-b+sqrt(d))/(2*a));}
        } else /*(a==0)*/ {
            if      (b==0&&c==0)  {return new Solution(true);}
            else if (b==0&&c!=0)  {return new Solution(false);}
            else  /*(b!=0&&c!=0)*/{return new Solution(-c/b);}
        }
    }
}

// модуль компиляции Solution.java
public class Solution {
    private int numberOfRoots;
    private double x1;
    private double x2;

    final static int ANY_NUMBER      = -1;
    final static int NOT_EXIST_ROOTS =  0;
    final static int ONE_ROOT        =  1;
    final static int TWO_ROOTS       =  2;

    public Solution (boolean anyRoots) {
        numberOfRoots=(anyRoots)?ANY_NUMBER:NOT_EXIST_ROOTS;
    }

    public Solution (double x1, double x2) {
        numberOfRoots=TWO_ROOTS;
        this.x1=x1;
```

```
        this.x2=x2;
    }

    public Solution (double x1) {
        numberOfRoots=ONE_ROOT;
        this.x1=x1;
    }

    public int getNumberOfRoots() {
        return numberOfRoots;
    }

    public double getX1() {
        if (numberOfRoots==ONE_ROOT||numberOfRoots==TWO_ROOTS) {
            return x1;
        } else {
            throw new Error("Sorry, X1 IS NOT EXIST OR ANY NUMBER.");
        }
    }

    public double getX2() {
        if (numberOfRoots==TWO_ROOTS) {
            return x2;
        } else {
            throw new Error("Sorry, X2 IS NOT EXIST OR ANY NUMBER.");
        }
    }
}

// модуль компиляции SolverOfSquareEquation.java
import java.io.*;
public class SolverOfSquareEquation {
    public static void main (String[] args) throws IOException {
        int numberOfRoots;
        double a, b, c, x1=0, x2=0;
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.print("a=");
        a=Double.parseDouble(br.readLine());
        System.out.print("b=");
        b=Double.parseDouble(br.readLine());
        System.out.print("c=");
        c=Double.parseDouble(br.readLine());
        SquareEquation sq=new SquareEquation(a, b, c);
        Solution sol=sq.solve();
        switch (sol.getNumberOfRoots()){
          case Solution.ANY_NUMBER:
               {System.out.println("X-любое число.");break;}
          case Solution.NOT_EXIST_ROOTS:
```

a	b	c	numberOfRoots	x1	x2
1	2	1	1	-1	–
1	-2	1	1	1	–
2	-7	6	2	1.5	2
1	0	-4	2	-2	2
0	0	0	∞	–	–
0	1	2	1	-2	–
0	0	1	0	–	–
0	1	0	1	0	–
2	-3.5	1	2	$\approx$0.35961	$\approx$1.39039

Таблица 1. Набор тестов для метода solve()

```
                {System.out.println("Корней нет.");break;}
            case Solution.ONE_ROOT:
                {System.out.println("X1="+sol.getX1());break;}
            case Solution.TWO_ROOTS:
                {System.out.println("X="+sol.getX1()+" X2="+sol.getX2());break;}
        }
    }
}
```

Класс Solution обзавелся тремя конструкторами для реализации принципа инкапсуляции. Первый конструктор имеет один аргумент типа double, через который передается значение корня в случае единственного решения. Второй конструктор имеет два параметра для передачи значений двух корней. И третий конструктор необходим для представления двух случаев: отсутствия корней и ситуации, когда корнем является любое число.

В Solution прописаны четыре статические константы, определяющие количество корней и используемые для повышения читаемости кода. Определены три метода доступа, позволяющие классу SolverOfSquareEquation отобразить полученные результаты. Для защиты пользователя класса от случайных ошибок во время разработки в методах доступа стоит проверка корректности операций.

4. Тестирование классов

При тестировании необходимо проверить работу методов классов. Автоматическое тестирование необходимо не столько для проверки правильности работы модулей, сколько для проведения последующего рефакторинга [2]. Кроме того, часто при создании и выполнении тестов становится понятно, что проектирование можно было выполнить еще проще.

Покажем, как это можно сделать на примере метода solve().

Составим таблицу тестов, проверяющую все варианты решения (табл. 1).

Воспользуемся инструментом JUnit (запускается из контекстного меню проекта, раздел Tools). Этот инструмент генерирует набор классов для тестирования классов нашего приложения. После генерации можем использовать семей-

ство набор assert-команд для сравнения ожидаемого результата с вычисленным. Но для правильной работы системы автоматического тестирования необходимо реализовать дополнительный метод equals(), унаследованный от Object.

Метод equals() используется командами типа assertEquals() для сравнения классов по значениям (по содержимому полей атрибутов, а не по ссылкам). Другими словами, equals() реализует сравнение идентичности объектов с точки зрения предметной области, в то время как оператор == с точки зрения программной реализации.

Если переписать подходящим образом метод toString() в классе Solution, также унаследованный от Object, то при ошибке тестирования можно будет увидеть не хэш-значения классов, на которых произошел сбой, а сами значения. Кроме того, можно хорошо сэкономить в SolverOfSquareEquation: код с оператором выбора switch переносится в toString(), а в классе Solution отпадает необходимость в реализации методов доступа и отслеживании корректности их вывода. Код становится более простым и прозрачным.

Последний вариант изменений приведен ниже.

```
// Изменения в Solution
public class Solution {
    ...
    // Описание атрибутов и конструкторов осталось без изменений
    // Константы можно сделать закрытыми
    // Удалены все методы доступа типа getX()
    // Для повышения читабельности кода вводим две константы
    public static boolean ROOT_IS_ANY_NUMBER=true;
    public static boolean ROOT_ISNT_EXIST=false;

@Override
    public boolean equals(Object obj) {
        double eps = 1e-5;
        if (obj == null) { return false; }
        if (getClass() != obj.getClass())  { return false; }
        final Solution other = (Solution) obj;
        if (this.numberOfRoots != other.numberOfRoots) { return false; }
        if (this.numberOfRoots==ONE_ROOT) {
               return abs(this.x1-other.x1)<eps
        }
        if(this.numberOfRoots==TWO_ROOTS){
               return abs(this.x1-other.x1)<eps && abs(this.x2-other.x2)<eps;
        }
        return true;
    }

    @Override
    public String toString() {
        String s=null;
          switch (getNumberOfRoots()){
```

```
            case Solution.ANY_NUMBER:
                 {s="X is any number.";break;}
            case Solution.NOT_EXIST_ROOTS:
                 {s="Roots not exist";break;}
            case Solution.ONE_ROOT:
                 {s="X="+x1;break;}
            case Solution.TWO_ROOTS:
                 {s="X1="+x1+" X2="+x2;break;}
        }
        return s;
    }
...
// Изменения в SolverOfSquareEquation
public class SolverOfSquareEquation {
    public static void main (String[] args) throws IOException {
        int numberOfRoots;
        double a, b, c;
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.print("a=");
        a=Double.parseDouble(br.readLine());
        System.out.print("b=");
        b=Double.parseDouble(br.readLine());
        System.out.print("c=");
        c=Double.parseDouble(br.readLine());
        System.out.println(new SquareEquation(a, b, c).solve());
    }
}
```

В этом варианте описание атрибутов и конструкторов осталось без изменений. Статические константы: ANY_NUMBER, ONE_ROOT, TWO_ROOTS, NOT_EXIST_ROOTS можно сделать закрытыми, окончательно скрыв внутреннее устройство класса. Удалены все методы доступа типа getX(), теперь в них нет необходимости — их действия заменяет один метод toString().

Далее приведен один из возможных способов реализации метода тестового класса.

```
    public void testSolve() {
        System.out.println("solve");
        SquareEquation[] instance =
            {new SquareEquation( 1, 2, 1),
             new SquareEquation( 1,-2, 1),
             new SquareEquation( 2,-7, 6),
             new SquareEquation( 1, 0,-4),
             new SquareEquation( 0, 0, 0),
             new SquareEquation( 0, 1, 2),
             new SquareEquation( 0, 0, 1),
             new SquareEquation( 0, 1, 0),
             new SquareEquation( 2,-3.5,1),
```

```
        };
    Solution[] expResult = {
        new Solution(-1),
        new Solution(1),
        new Solution(1.5,2),
        new Solution(-2,2),
        new Solution(Solution.ROOT_IS_ANY_NUMBER),
        new Solution(-2),
        new Solution(Solution.ROOT_ISNT_EXIST),
        new Solution(0),
        new Solution(0.35961,1.39039),
    };
    Solution result = null;
    for (int i=0;i<instance.length;i++) {
        result=instance[i].solve();
        assertEquals(expResult[i], result);
    }
}
```

Запуск теста должен производиться при каждом внесении изменения в код реализации метода solve(), для того чтобы убедиться в том, что функциональность не пострадала от внесенных в код поправок. В NetBeans это легко осуществить с помощью комбинации Alt+F6.

Последний вариант диаграммы классов приведен на рисунке 3.

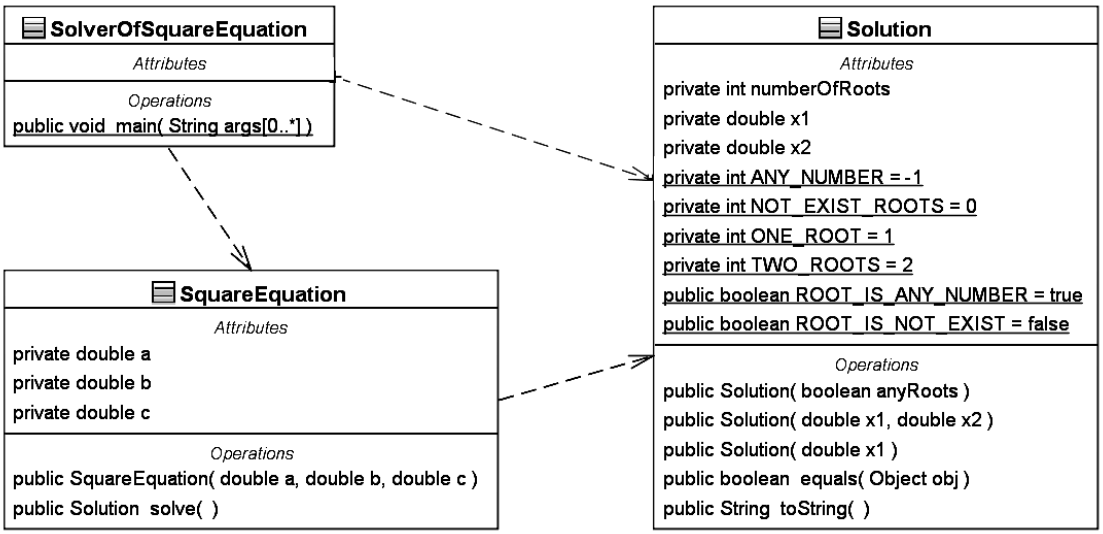

Рис. 3. Итоговая UML-диаграмма классов, представляющая архитектуру приложения SolverSquareEquation. Получена с помощью функции обратного реинженеринга в NetBeans

5. Заключение

Теперь пришло время взглянуть на диаграммы классов, которые были получены в процессе разработки (рисунки 1, 2, 3). Видим, что процесс проектирования — творческий процесс. Понятия предметной области превратились в классы и обросли обязанностями. Учащийся должен взглянуть еще раз на весь процесс, чтобы увидеть все промежуточные решения, увидеть цель — простой и ясный код, оптимальную с точки зрения нашей постановки задачи архитектуру. Необходимо еще раз указать на все преимущества и недостатки этих решений.

Наш пример позволяет сконцентрироваться на двух важных моментах: принципе инкапсуляции и автоматизации тестирования модулей. Кроме того, удалось отделить классы уровня интерфейса от классов уровня логики работы приложения [1]

Возможны и другие решения данной задачи, исходящие из других принципов проектирования. Не существует единственно верного проектного решения.

Можно продолжать развитие этого примера, постепенно усложняя задачу. Например, потребовать, чтобы наше приложение работало с любыми типами многочленов, тогда можно показать необходимость введения наследования и реализации полиморфного поведения классов нахождения корней. Интегрированная среда разработки NetBeans, имея встроенные средства рефакторинга, может помочь преобразовать наше приложение в расширяемый вид и в этом случае.

В заключение, хочу поблагодарить доцента кафедры информационной безопасности Надежду Федоровну Богаченко и ведущего программиста фирмы Luxsoft Максима Потанина, за обсуждение примера, приведенного в данной статье.

Литература

1. Ларман, К. Применение UML и шаблонов проектирования / К. Ларман. 2-е издание. – М.: Издательский дом «Вильямс», 2004. – 624 с.
2. Фаулер, М. Рефакторинг: улучшение существующего кода / М. Фаулер. – СПб: Символ-Плюс, 2003. – 432 с.
3. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, А. Джекобсон. – М.: ДМК Пресс. Питер, 2004. – 430 с.

Математические структуры и моделирование

Выпуск 20

Журнал

Редактор Е.В. Брусницына

Подписано в печать 20.09.2009.
ОП. Формат 60×84 1/8. Печ.л. _____ . Уч.-изд.л. _____.
Тираж 100 экз.

Отпечатано в _____
_____, г. Омск, _____, тел. _____
Лицензия _____