К вопросу о восстановлении трехмерного тела по плоским проекциям

Д. В. Алексеев

В работе рассматривается задача восстановления трехмерного изображения по кодам его плоских проекций. Описан алгоритм, работающий существенного быстрее ранее предложенного в [3]. Приведены результаты компьютерных экспериментов по сравнению скорости работы этих алгоритмов: старого и нового.

Ключевые слова: распознавание образов, аффинные преобразования, геометрическое распознавание, стереозрение.

1. Введение

В последнее время активно развивается направление, связанное с аффинным распознаванием образов. В работах [3]–[4] было показано, что класс аффинно-эквивалентных плоских изображений может быть однозначно задан с помощью кодов.

В этих же работах были получены результаты, позволяющие восстанавливать тело по его плоским проекциям. Эти результаты используются в образовательном процессе на кафедре МаТИС ([2]). Так, например, одним из стандартных заданий для студентов 3 курса (напр. в [6]) является упражнение на восстановление трехмерного тела по его плоским проекциям.

В разделе 2 приводится основной результат работы — теорема 1, и ее доказательство. Она дает возможность восстанавливать трехмерное изображение напрямую из кодов его плоских проекций, не восстанавливая сами эти проекции.

На основе вышеуказанной теоремы построен алгоритм восстановления трехмерного изображения по кодам его плоских проекций, ис-

пользующий в качестве входных данных только коды плоских проекций. Его отличие от алгоритма, описанного в [3] состоит в том, что предложенный в этой работе алгоритм, в большей степени ориентирован на компьютерную реализацию, в то время как оригинальный алгоритм более ориентирован на классические геометрические построения.

Оба алгоритма были реализованы в виде компьютерных программ на языке Python (листинги приводятся в приложениях А–Б). Были проведены компьютерные эксперименты с целью сравнения скорости работы алгоритмов. Описание компьютерной реализации алгоритмов, проверка их корректности и результаты экспериментов приводятся в разделе 3.

2. Основные результаты

Определение 1. Плоским изображением ([3]) будем называть конечное (не пустое) множество точек $\mathbb{R}^2$, не лежащих на двух параллельных прямых.

Определение 2. Объемным изображением будем называть конечное (не пустое) множество точек $\mathbb{R}^3$, не лежащих на двух параллельных плоскостях.

Определение 3. Плоские (объемные) изображения A и B называются аффинно эквивалентными или a-эквивалентными, если существует $\mathcal{T}$ — невырожденное аффинное преобразование плоскости (пространства), переводящее $A \to B$, то есть $\mathcal{T}(a_i) = b_i$, $i = 1, \ldots, N$.

Определение 4. Ориентированной площадью $S'(\triangle ABC)$ будем называть величину, равную площади треугольника $\triangle ABC$, если вершины A, B и C расположены против часовой стрелки и той же площади, взятой со знаком «минус», если вершины идут по часовой стрелке. Она может быть найдена как

$$S'(\triangle ABC) = \frac{1}{2} \det \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{pmatrix},$$

где (x_1, y_1) — координаты вершины A, (x_2, y_2) — координаты B и (x_3, y_3) — координаты C,

Определение 5. Кодом изображения $K^{(aff)}$ будем называть множество чисел $K^{(aff)}(A) = \left\{\rho_{i,j,k;l,m,n} = \frac{S'_{i,j,k}}{S'_{l,m,n}}\right\}$, если $S'_{l,m,n} \neq 0$. В случае, если $S'_{l,m,n} = 0$ считаем, что соответствующий элемент кода не определен.

Замечание 1. В случае компьютерной реализации алгоритма необязательно хранить в оперативной памяти все $(C_N^3)^2$ значений кодов. Достаточно хранить в C_N^3 отношений вида $\left\{\rho_{i,j,k;1,2,3} = \frac{S'_{i,j,k}}{S'_{1,2,3}} \,|\, 1 \leqslant i < j < k \leqslant N\right\}$ (при условии, что нумерация выбрана так, что точки с номерами 1,2,3 не лежат на одной прямой), а остальные вычислять по формуле $\rho_{i,j,k;l,m,n} = \frac{\rho_{i,j,k;1,2,3}}{\rho_{l,m,n;1,2,3}}$.

Замечание 2. В предлагаемом далее алгоритме используются только значения вида $\rho_{1,i,3;1,2,3}$ и $\rho_{1,2,i;1,2,3}$, где $i = 4, 5, \ldots, N$.

Как показано в [3] этот код является инвариантным относительно аффинных преобразований плоскости. Докажем вспомогательную лемму, позволяющую восстанавливать двумерное изображение по его коду.

Лемма 1. *Обозначим* $x_i = \rho_{i,2,3;1,2,3}(A)$, $y_i = \rho_{1,i,3;1,2,3}(A)$. *Тогда множество точек* $\tilde{A} = \{\tilde{a}_i(x_i, y_i) \,|\, i = 1, \ldots N\}$ *аффинно эквивалентно множеству* A.

Доказательство. Рассмотрим аффинное преобразование $\mathcal{T}$, которое переводит точки $\mathcal{T} : a_1 \to a'_1(0,0) = \tilde{a}_1$, $\mathcal{T} : a_2 \to a'_2(1,0) = \tilde{a}_2$ и $\mathcal{T} : a_3 \to a'_3(0,1) = \tilde{a}_3$. Точки a_1, a_2, a_3 не лежат на одной прямой, поэтому такое преобразование существует и единственно.

Зафиксируем произвольное $i > 3$, пусть преобразование переводит $\mathcal{T} : a_i \to a'_i$. Заметим, что преобразование $\mathcal{T}$ сохраняет отношение ориентированных площадей, следовательно, $\rho_{i,2,3;1,2,3}(A) = \frac{S'_{i,2,3}}{S'_{1,2,3}} = 2S'(\triangle a'_i \tilde{a}_2 \tilde{a}_3)$. Очевидно, что это условие определяет геометрическое место точек — прямую $x = \rho_{i,2,3;1,2,3}(A)$. Аналогично доказывается, что точка a'_i лежит на прямой $y = \rho_{1,i,3;1,2,3}(A)$. Следовательно, точка a'_i совпадает с $\tilde{a}_i$. Таким образом, изображение $\tilde{A}$ является образом A при аффинном отображении $\mathcal{T}$, что и требовалось доказать.

В [3] приводится геометрический алгоритм построения тела по его плоским проекциям. В случае точек общего положения он выглядит так:

Алгоритм 1. Классический алгоритм восстановления тела по его плоским проекциям, предложенный в работе [3].

Input: Заданы две проекции объемного изображения A вдоль неколлинеарных направлений l' и l'' на плоскости Π' и Π'', соответственно. Нумерация выбрана таким образом, что проекции точек a_1, a_2, a_3 не лежат на одной прямой.

ШАГ 1: Зададим точки $a_1 = a'_1$, $a_2 = a'_2$, $a_3 = a'_3$ и точку a_4 — произвольно (вне плоскости $(a_1a_2a_3)$).

ШАГ 2: Найдем аффинное отображение $\mathcal{T}$, которое переводит $a'_1 \to a''_1$, $a'_2 \to a''_2$, $a'_3 \to a''_3$.

for $i = 5 \to N$ **do**

ШАГ 3.1: Найдем $\tilde{a_4}$, $\tilde{a_i}$— образы точек a'_4 и a'_i при отображении T.

ШАГ 3.2: Найдем точку $\tilde{x}$ как пересечение прямых $\tilde{a_4}\tilde{a_i}$ и $a''_4a''_i$.

ШАГ 3.3: Найдем точку a_i, лежащую на прямой $\tilde{x}d$ и такую, что $\tilde{x}a_i : a_4a_i = \tilde{x}\tilde{a}_i : \tilde{a}_4\tilde{a}_i$.

end for

Output: Полученное тело $A = \{a_1, \ldots, a_N\}$ а-эквивалентно исходному.

Нижеприведенная теорема показывает, что возможно некоторое упрощение вышеприведенного алгоритма, адаптированное для компьютерной реализации.

Теорема 1. *Пусть задано трехмерное множество $\mathcal{A} = \{\alpha_1, \ldots, \alpha_N\}$. Пусть нумерация точек выбрана так, что проекции точек α_1, α_2 и α_3 на каждую из плоскостей Π' и Π'' не лежат на одной прямой, а точки α_1, α_2, α_2 и α_4 — не лежат в одной плоскости. Пусть $A^{(1)}$ и $A^{(2)}$ проекции множества $\mathcal{A}$ параллельно неколлинеарным направлениям l' и l'' на плоскость Π (не ограничивая общности можно считать, что они проектируются на одну и ту же плоскость). Пусть $\tilde{A}^{(1)}$ и $\tilde{A}^{(2)}$ — восстановленые по их кодам множества точек (см. лемму 1). Тогда*

1) *При всех $i > 3$ вектора $\overrightarrow{n}_i = \overrightarrow{\tilde{a}_i^{(1)}\tilde{a}_i^{(2)}}$ попарно коллинеарны.*
2) *Пусть $A^{(1)} = \{a_i^{(1)}(x_i, y_i) \,|\, i = 1, \ldots, N\}$. Обозначим $z_i = |\overrightarrow{n}_i|$, $i = 1, \ldots, N$. Тогда множество точек $\tilde{\mathcal{A}} = \{(x_i, y_i, z_i) \in \mathbb{R}^3 \,|\, i = 1, \ldots, N\}$ аффинно эквивалентно исходному множеству $\mathcal{A}$.*

Доказательство.
1. Рассмотрим аффинную систему координат, заданную следующим образом: α_1 — начало координат, $e_1 = \overrightarrow{\alpha_1\alpha_2}$, $e_2 = \overrightarrow{\alpha_1\alpha_3}$,$e_3 = \overrightarrow{\alpha_1\alpha_4}$ — вектора базиса. Пусть (ξ_i, η_i, ζ_i) — координаты точки α_i в этой системе координат, то есть $\overrightarrow{\alpha_1\alpha_i} = \xi_i \cdot e_1 + \eta_i \cdot e_2 + \zeta_i \cdot e_3$. Заметим, что указанное равенство сохранится при проектировании в A_1 и A_2 (так как сумма векторов и произведение вектора на число сохраняются при проектировании): $\mathcal{P}^{(j)}(\overrightarrow{\alpha_1\alpha_i}) = \xi_i \cdot \mathcal{P}^{(j)}(e_1) + \eta_i \cdot \mathcal{P}^{(j)}(e_2) + \zeta_i \cdot \mathcal{P}^{(j)}(e_3)$, $j = 1, 2$.

При аффинных преобразованиях указанные равенства тоже сохраняются, следовательно,

$$\overrightarrow{\tilde{a}_1^{(j)}\tilde{a}_i^{(j)}}) = \xi_i \cdot e_1^{(j)} + \eta_i \cdot e_2^{(j)} + \zeta_i \cdot e_3^{(j)},\ j = 1, 2,$$

где $e_k^{(j)}$ — образ вектора e_k при последовательном выполнении проектирования и аффинного преобразования, переводящего проекцию в $\tilde{A}^{(j)}$. Заметим, что по построению этого аффинного преобразования в доказательстве леммы 1 получается, что $\tilde{a}_1^{(j)} = (0, 0)$ — начало координат, а $e_1^{(j)} = (1, 0)$, $e_2^{(j)} = (0, 1)$ — вектора единичного репера. Обозначим $e_3^{(j)} = (p_1^{(j)}, p_2^{(j)})$. Координаты точки $\tilde{a}_i^{(j)}$ можно найти как

$$\tilde{a}_i^{(j)} = (\xi_i + \zeta_i \cdot p_1^{(j)}, \eta_i + \zeta_i \cdot p_1^{(j)}). \tag{1}$$

Следовательно,

$$\overrightarrow{\tilde{a}_i^{(1)}\tilde{a}_i^{(2)}} = \left(\zeta_i(p_1^{(2)} - p_1^{(1)}), \zeta_i(p_2^{(2)} - p_2^{(1)})\right) = \zeta_i \cdot \overrightarrow{p_0}, \tag{2}$$

то есть все вектора $\overrightarrow{n}_i$ коллинеарны вектору $\overrightarrow{p_0} = \left(p_1^{(2)} - p_1^{(1)}, p_2^{(2)} - p_2^{(1)}\right)$. Первое утверждение теоремы доказано.
2. Рассмотрим аффинное преобразование $\mathcal{T} : \mathbb{R}^3 \to \mathbb{R}^3$, переводящее точки α_1, α_2, α_3 и α_4 в точки $\tilde{\alpha}_1(0, 0, 0)$, $\tilde{\alpha}_2(1, 0, 0)$, $\tilde{\alpha}_3(0, 1, 0)$ и $\tilde{\alpha}_4(0, 0, 1)$, соответственно. Указанные точки не лежат в одной плоскости, следовательно, такое преобразование существует и единственно.

Зафиксируем $i > 3$ и рассмотрим $\tilde{\alpha}_i = \mathcal{T}(\alpha_i)$ — образ α_i при указанном преобразовании. Его координаты, очевидно, равны (ξ_i, η_i, ζ_i) — координатам в аффинной системе координат из первой части дока-

зательства. Применим к полученному изображению следующее аффинное преобразование:

$$\begin{cases} \xi' = & x + p_1^{(1)} \cdot \xi \\ \eta' = & y + p_2^{(1)} \cdot \eta \\ \zeta' = & |\overrightarrow{p_0}| \cdot \zeta \end{cases}$$

Из (1) следует, что координаты (ξ_i', η_i') совпадают с координатами точки $\tilde{a}_i^{(1)}$. Кроме того, из (2) следует, что $\overrightarrow{n}_i = \zeta_i \cdot \overrightarrow{p_0}$, поэтому $\zeta_i' = |\overrightarrow{p_0}| \cdot \zeta_i = |\overrightarrow{n}_i|$. Таким образом, полученное изображение совпадает с $\tilde{\mathcal{A}}$, следовательно, $\tilde{\mathcal{A}}$ аффинно эквивалентно исходному изображению.

Следствие 1. *Зададим произвольный вектор* $\overrightarrow{v} = (v_x, v_y)$. *Тогда множество точек с координатами* $b_i = (x_i + v_x \cdot z_i, y_i + v_y \cdot z_i)$ *аффинно эквивалентно некоторой проекции исходного тела.*

Доказательство. В качестве плоскости проектирования выберем $(\alpha_1\alpha_2\alpha_3)$ и введем на этой плоскости аффинную систему координат, в которой эти точки обладают следующими координатами: $\alpha_1(0,0)$, $\alpha_2(1,0)$ и $\alpha_3(0,1)$. Выберем направление проектирования таким образом, чтобы вектор $\overrightarrow{\alpha_1\alpha_4}$ проектировался в $\overrightarrow{v}$.

Тогда проекция вектора $\overrightarrow{\alpha_1\alpha_i} = x_i \cdot \overrightarrow{\alpha_1\alpha_2} + y_i \cdot \overrightarrow{\alpha_1\alpha_3} + z_i \cdot \overrightarrow{\alpha_1\alpha_4}$ в этой системе будет иметь координаты $b_i = (x_i + v_x \cdot z_i, y_i + v_y \cdot z_i)$. Очевидно, существует аффинное преобразование, переводящее эти точки в точки с такими же координатами, но уже в прямоугольной системе координат.

Указанное следствие может быть использовано для имитации вращения восстановленного тела, например, удобно задать $\overrightarrow{v} = (\cos\phi \cdot \sin\theta, \sin\phi \cdot \sin\theta)$.

Замечание 3. Если заранее неизвестно попарное соответствие точек двух проекции, то есть нумерация не задана, то первый пункт теоремы позволяет существенно сократить перебор вариантов. Действительно, если при построении какие-то два вектора $\overrightarrow{n}_{i_1}$ и $\overrightarrow{n}_{i_2}$ оказались неколлинеарны, то такой вариант не соответствует никакому объемному изображению и его можно отбросить.

Замечание 4. При компьютерной реализации алгоритма, в случае, когда $p_1^{(2)} - p_1^{(1)} \neq 0$, в качестве третьей координаты z_i вместо длины

вектора $|\overrightarrow{n}_i|$ можно использовать $z_i = \tilde{x}_i^{(2)} - \tilde{x}_i^{(1)}$ — то есть первую координату этого вектора. Если же $p_1^{(2)} - p_1^{(1)} = 0$, то, аналогично, можно использовать его вторую координату: $z_i = \tilde{y}_i^{(2)} - \tilde{y}_i^{(1)}$.

Ниже приводится алгоритм 2, построенный на основе теоремы 1.

Алгоритм 2. Новый алгоритм восстановления трехмерного изображения по кодам его плоских проекций, предложенный в данной работе.

Input: Заданы две проекции объемного изображения вдоль неколлинеарных направлений в виде кодов $\rho'_{i,j,k;l,m,n}$ и $\rho''_{i,j,k;l,m,n}$. Нумерация выбрана таким образом, что точки a_1, a_2, a_3 не лежат на одной прямой

ШАГ 1: $(x_1, y_1) = (0, 0)$, $(x_2, y_2) = (1, 0)$, $(x_3, y_3) = (0, 1)$

for $i = 1 \to N$ **do**

ШАГ 2.1: $(x_i, y_i) = (\rho'_{1,i,3;1,2,3}, \rho'_{1,2,i;1,2,3})$

if $\rho'_{1,2,4;1,2,3} = \rho''_{1,2,4;1,2,3}$ **then**

ШАГ 2.2: $z_i = \rho''_{1,i,3;1,2,3} - \rho'_{1,i,3;1,2,3}$

else

ШАГ 2.2: $z_i = \rho''_{1,2,i;1,2,3} - \rho'_{1,2,i;1,2,3}$

end if

end for

Output: Полученное тело $T = \{(x_1, y_1, z_1), \ldots, (x_N, y_N, z_N)\}$ а-эквивалентно исходному.

3. Компьютерная реализация и тестирование

Для сравнения скорости работы вышеприведенных алгоритмов была написана программа на языке Python (v.2.7) с использованием математической библиотеки NumPy (v.1.7.0). Приводится часть листинга программы, соответствующая реализации алгоритмов: как классического ([6]) — 1, так и нового — 2.

Тестирование проводилось на компьютере с процессором AMD Phenom×4 с тактовой частотой 3.0GHz. Предварительно было порождено два набора состоящих из 100 и 1000 случайно выбранных точек (листинг программы порождения точек приводится в приложении B). Были построены проекции вдоль двух случайно выбранных

направлений и сохранены в файлы. Алгоритмы (классический и новый) считывали точки и восстанавливали трехмерное изображение. Учитывалось только время работы алгоритма, время на чтение точек из файла не учитывалось. Каждый алгоритм повторялся 10 раз и рассматривалось среднее время работы.

Результаты работы алгоритмов приводятся в таблице 1.

Число точек	**Классический**		**Новый**	
	Общее время	Среднее	Общее время	Среднее
100	0.2294	0.02294	0.005926	0.0005926
1000	2.413	0.2413	0.06052	0.006052

Таблица 1. Время работы алгоритмов.

Видно, что новый алгоритм по скорости работы существенно превосходит классический.

Замечание 5. Если посмотреть листинги программ, то можно заметить, что оба алгоритма (классический и новый) работают с координатами точек, заданными в файле. Если использовать в качестве входных данных не координаты, а коды изображений, то скорость работы нового алгоритма должна возрасти, что дает еще большее превосходство над классическим алгоритмом.

Отдельно была проведена проверка корретности реализации обоих алгоритмов. На этапе порождения случайных точек в файл записывались исходные координаты трехмерного тела. После восстановления по проекциям рассматривалось трехмерное аффинное преобразование, переводящее первые 4 точки исходного объемного изображения $a_1, \ldots, a_4$ в соответствующие точки восстановленного изображения $\tilde{a}_1, \ldots, \tilde{a}_4$. После этого осуществлялась проверка того, что остальные точки исходного тела a_i, $i = 5, \ldots, N$ переходят в соответствующие им точки восстановленного изображения $\tilde{a}_i$ (при найденном преобразовании). Проверка показала, что оба алгоритма корректно работают на массивах из 100 и 1000 случайных точек. Листинг основной части программы, осуществляющей вышеуказанную проверку корректности, приводится в приложении Г.

4. Выводы

Указанный алгоритм может быть использован как в задачах восстановления стереоизображений, так и в других задачах распознавания и построения трехмерных объектов.

Кроме того, указанный алгоритм может быть использован в курсе «Дискретный анализ и интеллектуальные системы». Как показала практика[1], при решении задач, связанных с восстановлением трехмерных изображений по их плоским проекциям, студенты допускают множество арифметических ошибок. Вышеприведенный алгоритм позволяет существенно сократить количество вычислений.

Список литературы

[1] Александров П. С. Курс аналитической геометрии и линейной алгебры: Учебник. 2-е изд. — СПб.: Лань, 2009.

[2] Алексеев Д. В. Использование метода В. Н. Козлова в образовательном процессе на кафедре МаТИС // Интеллектуальные системы. — 2013. Т. 17, вып. 1–4. — С. 16–20.

[3] Козлов В. Н. Элементы математической теории зрительного восприятия. — М.: Изд-во ЦПИ при мех.-мат. ф-те МГУ, 2001.

[4] Козлов В. Н. Доказательность и эвристика при распознавании визуальных образов // Интеллектуальные системы. — 2010. Т. 14, вып. 1–4. — С. 35–52.

[5] Kozlov V. N. Image Coding and Recognition and Some Problems of Stereovision // Pattern Recognition and Image Analysis. — 1997. Vol. 7, No. 4. — P. 448–466.

[6] Кудрявцев В. Б., Гасанов Э. Э., Подколзин А. С. Введение в теорию интеллектуальных систем. М.: Изд-во ф-та ВМиК МГУ, 2006.

Приложение А. Стандартный алгоритм

```
from numpy import matrix, array
from scipy.linalg import det, solve
```

[1] Автор статьи более 10 лет ведет этот спецсеминар

```
def restorePointStdOpt(P1, P2, pt1, pt2, T):
    '''Restores a point using standard algorithm
    T is a precomputed affine transform
    T: P1[i]->P2[i], i=0,1,2'''
    #T = find_affine_transform(P1[0:3], P2[0:3])
    D = P2[3]
    E = pt2
    Dtilda = apply_transform(T, P1[3])
    Etilda = apply_transform(T, pt1)
    X = find_intersection([D, E], [Dtilda, Etilda])
    z = find_proportion(D, E, X)
    x,y = pt1[0], pt1[1]
    return x, y, z

def find_proportion(D, E, X):
    '''return |EX|:|DX|'''
    if D[0] != X[0]:
        return (E[0]-X[0])/(D[0] - X[0])
    else:
        return (E[1]-X[1])/(D[1] - X[1])

def find_affine_transform(P, Q):
    '''Finds affine transform that:
    p0->q0, p1->q1, p2->q2
    Returns  transform coefficients in form
    [a11, a12, a21, a22, b1, b2]'''
    def mkMatr(X): # X - list
        return  [X+ [0,0,1,0], [0,0]+X+[0,1] ]
    M0, M1, M2 = mkMatr(P[0]), mkMatr(P[1]), mkMatr(P[2])
    MM = [M0[0], M0[1], M1[0], M1[1], M2[0], M2[1]]
    M = matrix(MM)
    b = matrix(Q[0]+Q[1]+Q[2])
    T = solve(M,b.T)
    return T

def apply_transform(T, x):
    x_new=T[0,0]*x[0]+T[1,0]*x[1]+T[4,0]
    y_new=T[2,0]*x[0]+T[3,0]*x[1]+T[5,0]
    return [x_new, y_new]

def find_intersection(L1, L2):
    '''Find intersection of lines,
```

```
    each is defined by 2 points L[0] and L[1] '''
    C1  = find_line_equation(L1)
    C2  = find_line_equation(L2)
    M  = matrix([C1[:2] , C2[:2]])
    b  = array([C1[2], C2[2]])
    return solve(M, b)

def find_line_equation(L):
    '''Find equation of the line
    that goes through points L[0] and L[1]
    for equation   a*x+b*y=c returns [a,b,c]'''
    napr = [L[1][0] - L[0][0], L[1][1] - L[0][1]]
    normal_dir = [ napr[1], -napr[0] ]
    c = normal_dir[0] * L[1][0] +   normal_dir[1] * L
        [1][1]
    return normal_dir + [c,]
```

Приложение Б. Новый алгоритм

```
def restorePointNew(P1, P2, pt1, pt2, useY, SP1, SP2):
    '''Restores point 3D coordinates,
    using basis points P1 and P2,
    point projections p1 and p2'''
    x = S([P1[0],pt1 , P1[2]]) /SP1
    y = S([P1[0],P1[1], pt1])  /SP1

    if useY:
        z = S([P2[0],P2[1], pt2]) / SP2 - y
    else:
        z = S([P2[0], pt2, P2[2]])/ SP2 - x
    return x,y,z

def Code(P, Q):       # P&Q -- 3-point arrays
    return S(P)/S(Q)

def S(P):
    """Oriented  area of triangle P[0], P[1], P[2] (x2)
       """
    s=  P[0][0]*P[1][1] + P[1][0]*P[2][1] + P[2][0]*P
        [0][1] \
      - P[0][1]*P[1][0] - P[1][1]*P[2][0] - P[2][1]*P
          [0][0]
    return s
```

Приложение В. Порождение случайных точек

```
from random import random, uniform
from math import cos, sin, pi
from sys import argv

def usage():
    print 'Usage:'
    print argv[0], 'NumPt Filename'
    print 'NumPt - number of random points to generate'
    print 'Filename - to write output to'

def main():
    if len(argv) < 3: usage()
    else:
        num_pt=int(argv[1])
        file_name=argv[2]
        generatePoints(num_pt, file_name)

def generatePoints(num_pt, file_name):
    """Generates num_pt random points
    and write their projections to file <<file_name>>"""
    dir1, dir2 = getRandomDirection(), getRandomDirection
        ()
    with open(file_name, 'w') as f:
        for i in range(num_pt):
            point = getRandomPoint()
            pt1 = getProjection(point, dir1)
            pt2 = getProjection(point, dir2)
            print >> f, pt1[0], pt1[1], pt2[0], pt2[1],\
                point[0], point[1], point[2]

def getRandomPoint():
    return [ random(), random(), random() ]

def getRandomDirection():
    phi = uniform(0, 2*pi)
    theta= uniform(pi/8, pi/2-pi/8)
    #not too close to 0 or pi/2
    x,y,z = cos(theta)*cos(phi), cos(theta)*sin(phi), sin
        (theta)
    return [x,y,z]
```

```
def getProjection(point, dir):
    lam= point[2] / dir[2]
    x, y = point[0]+lam*dir[0],   point[1]+lam*dir[1]
    return [x,y]

if __name__=='__main__':
    main()
```

Приложение Г. Проверка корректности работы алгоритма восстановления

```
from numpy import matrix,array
from scipy.linalg import det, solve
from CompareRestore3D import read_points, restore_points

def test_algorithm_correctness(filename, alg_type):
    """Tests correct restoration of 3D points
    From <filename> using alg_type algorithm"""
    P1, P2, pointsOrig = read_points(filename)
    pointsRestored = restore_points(P1, P2, alg_type)
    if are_affine_equivalent_3D(pointsOrig,
       pointsRestored):
        print 'OK, test passed'
    else:
        print 'ERROR: test failed!'

def are_affine_equivalent_3D(P1, P2):
    """Compares two arrays of 3d points
    for being affine-equivalent
    Returns boolean"""
    def distance3(p,q):
        x, y, z=p[0]-q[0], p[1]-q[1], p[2]-q[2]
        return x*x + y*y + z*z

    epsilon = 1.0e-5
    reper=[ [0,0,0], [1,0,0], [0,1,0], [0,0,1]]
    T1 = find_affine_trans_3D(P1[0:4], reper)
    T2 = find_affine_trans_3D(P2[0:4], reper)
    for pt1, pt2 in zip(P1, P2):
        pt11 = apply_transform_3D(T1, pt1)
        pt22 = apply_transform_3D(T2, pt2)
        if distance3(pt22, pt11) > epsilon:
            return False
```

```
    return True

def apply_transform_3D(T, x):
    """Transform␣is␣a␣list␣of␣format:
␣␣␣␣T=[␣a11,␣a12,␣a13,
␣␣␣␣␣␣␣␣a21,␣a22,␣a23,
␣␣␣␣␣␣␣␣a31,␣a32,␣a33,
␣␣␣␣␣␣␣␣b1,␣b2,␣b3
␣␣␣␣␣␣]"""
    x_new=T[0,0]*x[0]+T[1,0]*x[1]+T[2,0]*x[2]+T[9,0]
    y_new=T[3,0]*x[0]+T[4,0]*x[1]+T[5,0]*x[2]+T[10,0]
    z_new=T[6,0]*x[0]+T[7,0]*x[1]+T[8,0]*x[2]+T[11,0]
    return [x_new, y_new, z_new]

def find_affine_trans_3D(P, Q):
    """Finds␣3D␣affine␣transform␣that:
␣␣␣␣␣␣␣␣p0->q0,␣p1->q1,␣p2->q2,␣p3->q3
␣␣␣␣Returns␣␣transform␣coefficients␣as␣a␣list␣in␣form
␣␣␣␣T=[␣a11,␣a12,␣a13,
␣␣␣␣␣␣␣␣a21,␣a22,␣a23,
␣␣␣␣␣␣␣␣a31,␣a32,␣a33,
␣␣␣␣␣␣␣␣b1,␣b2,␣b3
␣␣␣␣␣␣]"""
    def mkMat3(X): # X - list of 3
        return  [ X+ [0,0,0,0,0,0,1,0,0],\
                  [0,0, 0]+X+[0,0,0,0,1,0],\
                  [0,0, 0, 0,0,0]+X+[0,0,1] ]
    M0, M1, M2, M3 = mkMat3(P[0]), mkMat3(P[1]), \
                     mkMat3(P[2]) , mkMat3(P[3])
    MM = [  M0[0], M0[1], M0[2], \
            M1[0], M1[1], M1[2], \
            M2[0], M2[1], M2[2], \
            M3[0], M3[1], M3[2]]
    M = matrix(MM)
    b = matrix([Q[0]+Q[1]+Q[2]+Q[3]])
    T = solve(M,b.T)
    return T
```