Modeling and recognition of 2D/3D images

3D-реконструкция по двум изображениям (3D reconstruction from two images)

Автор: sv | 05.02.2018 Нет комментариев

Введение
Формализация задачи
Алгоритм 3D-реконструкции
Простейшая модель для тестирования алгоритма
Создание фото при помощи программы 3D GL WinApi
Подготовка программы для тестирования алгоритма
Тестирование алгоритма с реальным объектом
Контрольные задания
Выводы
Полезные ссылки

Введение

Трехмерная реконструкция (англ. 3D reconstruction) – это процесс получения 3D объекта на основе изображений (см. статью Фотограмметрия). На вход алгоритма обработки подается набор из нескольких изображений (два или более), результатом работы которого является 3D фотография (рис.1).

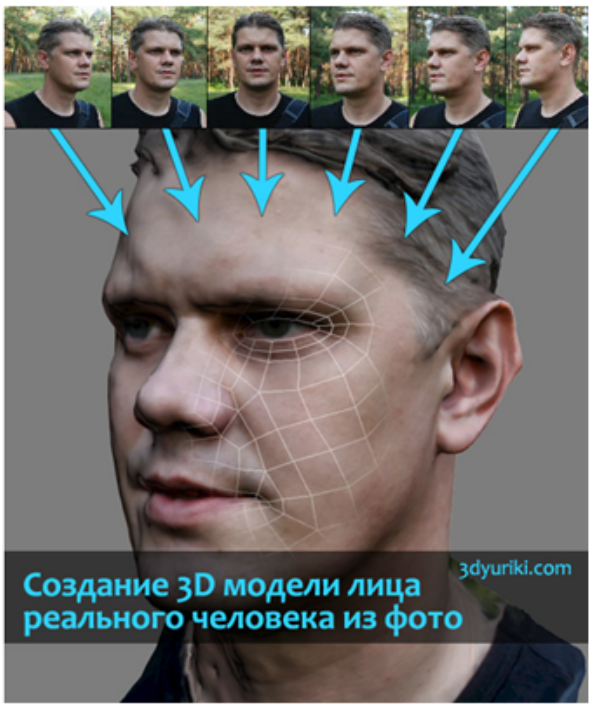


Рис. 1. Создание 3D модели лица человека из серии фотографий

Известно, что по двум или более изображениям (проекциям) можно восстановить пространственный объект (рис.2). Для этого достаточно установить соответствие между точками изображений.

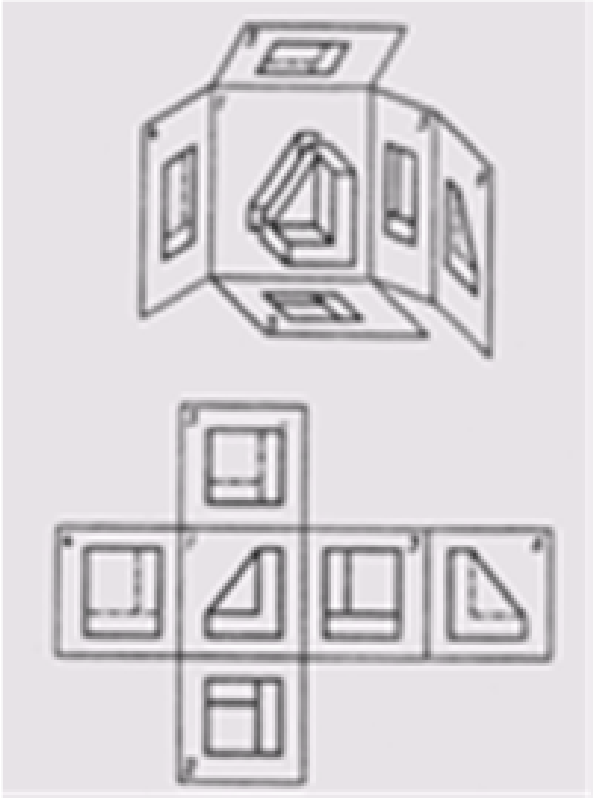


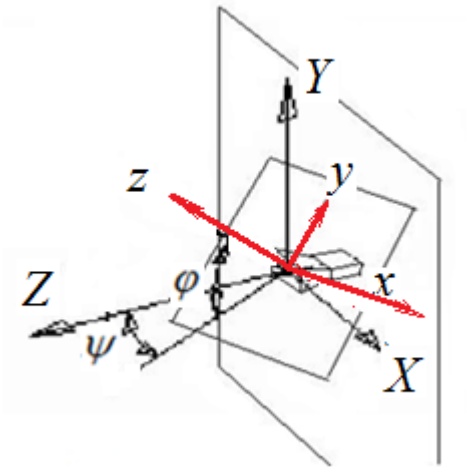
Рис. 2. Реконструкция трехмерного объекта по техническому чертежу

В чертежах такая зависимость устанавливается на основе визуального восприятия. Попробуем выразить эту зависимость математически.

Формализация задачи

Суть математического решения задачи 3D-реконструкции по изображениям заключается в определении зависимостей между точками 2-х проекций.

При выполнении контрольных заданий № 7 и №8 из темы 3D графика на основе WinApi C++ был продемонстрирован переход от главного вида антенны к вспомогательному за счет создания новой (вспомогательной) СК.



Ориентация новой СК определялась при помощи матрицы обратного преобразования

$$B^{-1} = R_y^{-1} R_x^{-1} = \begin{pmatrix} \cos\psi & \sin\psi\sin\varphi & \sin\psi\cos\varphi & 0 \\ 0 & \cos\varphi & -\sin\varphi & 0 \\ -\sin\psi & \cos\psi\sin\varphi & \cos\psi\cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Эту же матрицу можно использовать для установления зависимости между главной и вспомогательной системами координат.

$$(x\ y\ z\ 1) = (X\ Y\ Z\ 1)\, B^{-1}$$

В итоге получаем систему уравнений для перехода от координат точек пространственного объекта, определенного в главной СК *XYZ*, к координатам точек на плоскости *xy* вспомогательной СК:

$$x = X\cos\psi - Z\sin\psi$$

$$y = X\sin\psi\sin\varphi + Y\cos\varphi + Z\cos\psi\sin\varphi$$

$$z = 0$$

Алгоритм 3D-реконструкции

Полученная выше система уравнений позволяет найти координату *Z* каждой точки (*X,Y*) на главном изображении, если известны углы между СК и координаты соответствующей точки (*x,y*) на вспомогательном изображении. Соответствие между точками может устанавливаться по их цвету.

Краткое описание алгоритма:

- Для каждой точки главного изображения (X,Y) определяем соответствующую точку на вспомогательном изображении (x,y), осуществляя итерацию координаты Z
- Соответствующая точка найдена, если совпадают 3 цвета точек (RGB) на обоих изображениях.
- Вместе с соответствующей точкой определяется и координата Z.

Детальное описание алгоритма:

Для поиска соответствующей точки на двух изображениях необходимо в итерационном режиме:

- перевести координаты точки из пространства фото (*I,J*) в пространство логических координат (*XY*)
- задать координату *Z* (*Z=Z+dZ*)
- выполнить пересчет координат точки из главной СК (*X,Y,Z*) во вспомогательную СК (*x,y*)
- перевести координаты точки из пространства логических координат (*x,y*) в пространства фото (*I1,J1*)
- проверить на соответствие по цвету точки главного изображения (*I,J*) и вспомогательного (*I1,J1*)

Цифровое изображение хранится в виде прямоугольной матрицы, элементы которой несут информацию о цвете элементарных участков изображения (пикселов), а номера строки I и столбца J элемента определяют его положение в матрице.

Различают две прямоугольных системы координат цифрового изображения (рис.7):

- началом которой является левый нижний пиксел;
- началом которой является левый верхний пиксел.

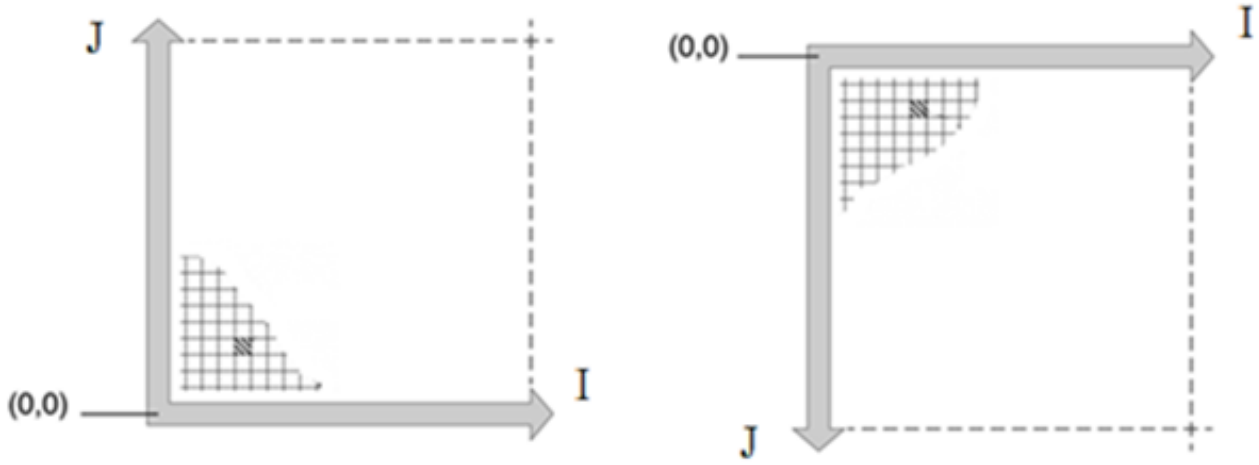


Рис. 7. Две разновидности прямоугольных системы координат цифрового изображения

Первая СК (на рис. слева) принята при записи изображений в файл во всех форматах (включая и формат BMP файла).

В фотограмметрии традиционно применяется вторая СК (на рис. справа). В оконных WinAPI приложениях также используют эту систему координат.

Для перехода из одной СК в другую достаточно выполнить преобразование *J = H — J*.

В логической системе координат (рис.8) вершины нормализуются, т.е. удовлетворяют следующим условиям:

- центр координат перенесен в центр экрана;
- направление оси Y вверх;
- координаты (от -1 до +1) соотнесены с шириной (Width) и высотой окна (Height) .

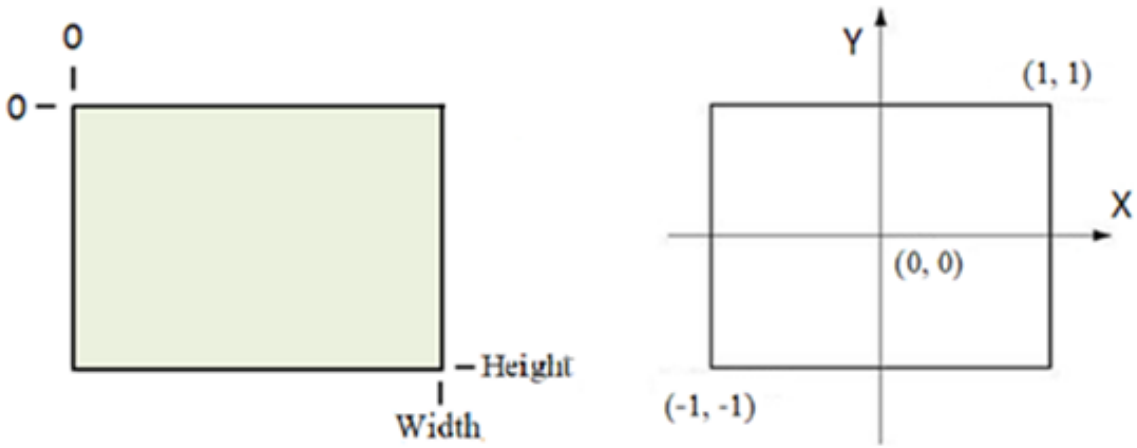


Рис. 8. Переход от цифровой к логической системе координат

Ниже приведены соотношения для перехода из СК цифрового изображения в нормализованную СК и обратно.

```
X= 2 * I / Width  - 1
Y= 1 -  2 * J / Height         для левой СК       Y= 2 * J / Height -1
0 < Z < 1
I1 = Width * (x + 1) / 2
J1 = - Height * (y - 1) / 2    для левой СК       J1= Height * (y - 1) / 2
```

Определяются разницы сигналов между предполагаемыми соответствующими точками:

```
dR = abs(rgb1[I1][ J1].rgbRed - rgb[I][ J].rgbRed);
dG = abs(rgb1[I1][ J1].rgbGreen - rgb[I][ J].rgbGreen);
dB = abs(rgb1[I1][ J1].rgbBlue - rgb[I][ J].rgbBlue);
```

Соответствие между точками установлено, если выполняется комбинированное условие:

```
(dR < dmin и dG < dmin и dB < dmin)
```

При выполнении этого условия одновременно определяется и значение координаты Z искомой точки.

Простейшая модель для тестирование алгоритма

Рассмотренный выше алгоритм 3D реконструкции по изображениям применим для случая, когда известно взаимное положение камер в системе 3D видеонаблюдения.

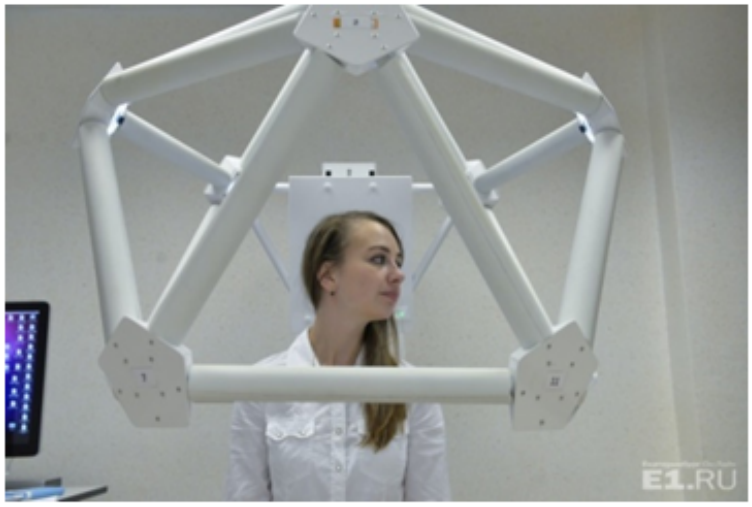


Рис. 9. Система 3D видеонаблюдения

Этот алгоритм можно протестировать через 3D-реконструкцию цветного треугольника на основе 2-х фотографий , полученных под разными углами камеры.

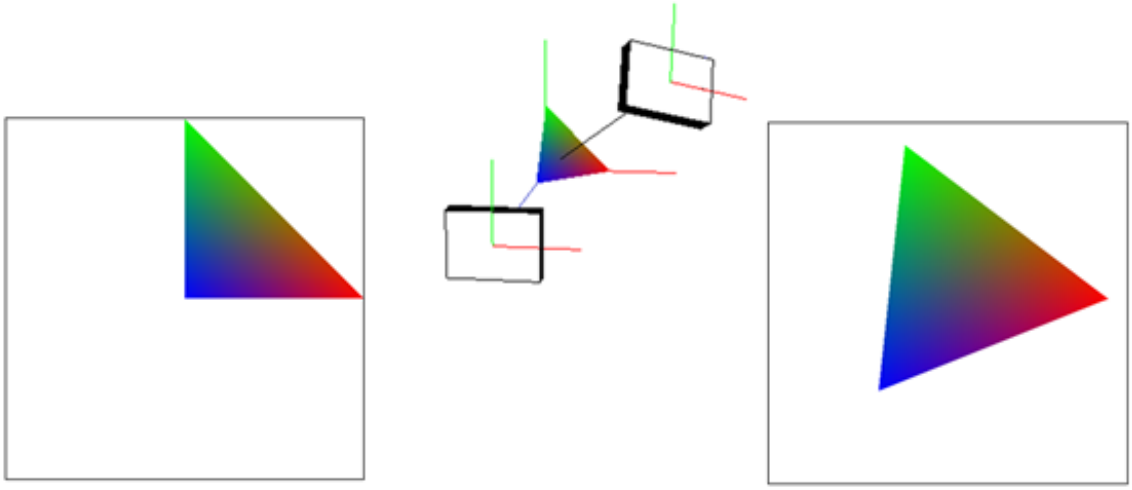
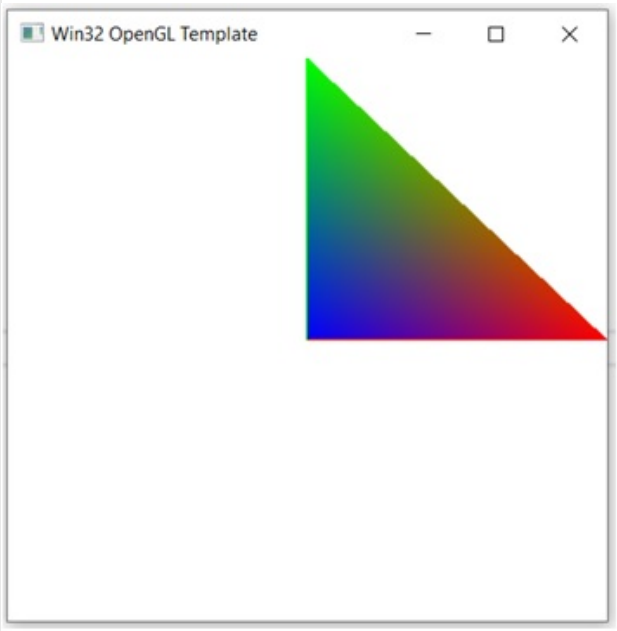
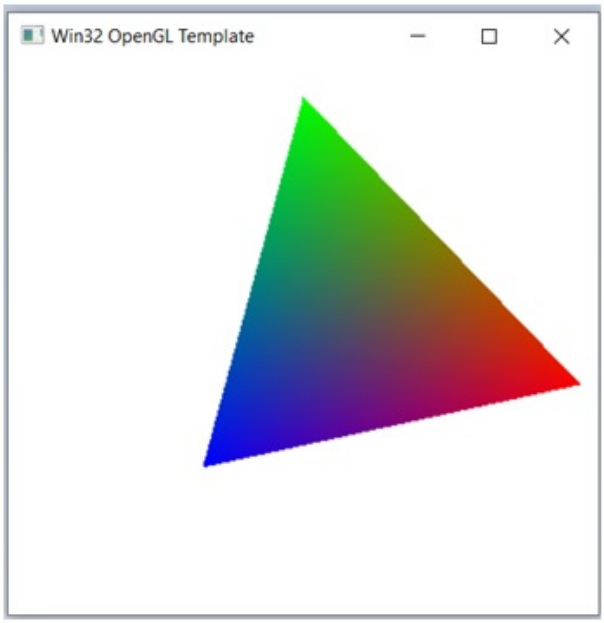


Рис. 5. Подготовка изображений для тестирования алгоритма задачи 3D реконструкции

Создание фото при помощи программы 3D_GL WinApi

Для создания фото воспользуемся программой-шаблоном из статьи 3D графика на основе OpenGL WinApi C++. Ее можно скачать с моего Googl-диска по ссылке 3d_gl.zip. Модифицируем программу-шаблон, чтобы получить цветные фото треугольника под разными направлениями взгляда.





Для этого:

- В файле main.cpp (функция WinMain) при создании окна устанавливаем его размеры 500*500.

```
RegisterClass(&wcl);
 hWnd = CreateWindow((LPCWSTR) "OpenGLWinClass", L"Win32 OpenGL Template",
 WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN | WS_CLIPSIBLINGS,
// Размеры окна выбираются с учетом отступов (отсекаются при вырезании фото)
 200, 150, // Позиция левого верхнего угла
 500, 500, // Ширина и высота планируемого размера фото (500* 500)
 HWND_DESKTOP, NULL,
 hThisInst, NULL);
```

- В файле engine.cpp вместо синего фона устанавливаем белый:

```
GLvoid Engine::Init(GLvoid) {
 //glClearColor(0.85f, 0.85f, 1.0f, 1.0f); // Устанавливается голубой фон
 glClearColor(1.0f, 1.0f, 1.0f, 1.0f); // Устанавливается белый фон
```

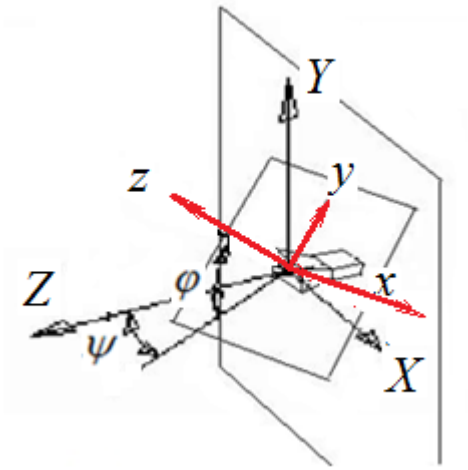
- Устанавливаем ортогональную проекцию вместо перспективной и изменяем точку направления взгляда

```
GLvoid Engine::Draw(GLvoid)
{
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Очищается буфер кадра и буфер глубины
 // Текущая матрица сбрасывается на единичную
 glMatrixMode(GL_PROJECTION); // Действия будут производиться с матрицей проекции
 glLoadIdentity();
 //glTranslatef(0.0, 0.5, 0.0);
 //glFrustum(-1, 1, -1, 1, 3, 10); // Устанавливается перспективная проекция
 //gluPerspective(30, 1, 3, -3); // Устанавливается перспективная проекция
 glOrtho(-1, 1, -1, 1, 3, 10); // Устанавливается ортогональная проекция
 //gluOrtho2D(-10, 10, -10, 10); // Устанавливается ортогональная проекция
 glMatrixMode(GL_MODELVIEW); // Действия будут производиться с матрицей модели
 glLoadIdentity();
 gluLookAt(0, 0, 5, 0, 0, 0, 0, 1, 0); //Определяется вид по точке (0, 0, 5)
//gluLookAt(1.48, 2.5, 4.07, 0, 0, 0, 0, 1, 0); //Определяется вид по точке ((1.48, 2.5, 4.07)

//DrawAxes(); //рисуются оси камеры
```

- Добавляем в функцию *DrawModel* модель треугольника и комментируем все остальное:

```
GLvoid Engine::DrawModel(GLvoid) // Отрисовка модели
{
 glBegin(GL_TRIANGLES);
 // v0-v1-v2
 glColor3d(1, 0, 0);
 glVertex3f(1.0, 0.0, 0.0);
 glColor3d(0, 1, 0);
 glVertex3f(0.0, 1.0, 0.0);
 glColor3d(0, 0, 1);
 glVertex3f(0.0, 0.0, 1.0);
 glEnd();
 //glColor3d(0, 0, 0);
 //glBegin(GL_LINE_LOOP);
 //// v0-v1-v2-v3
 //glVertex3f(0.4, 0.2, 0);
```

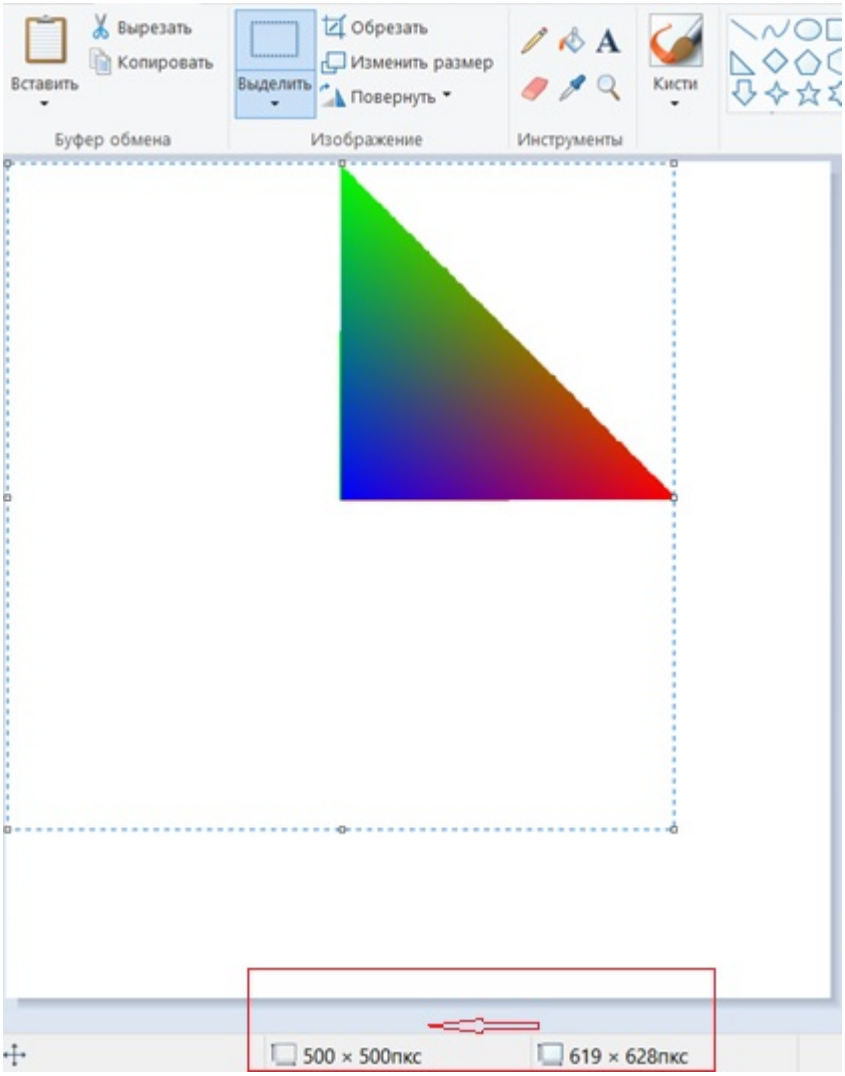
Для второго изображения координаты точки вида определяются, по следующим зависимостям:




```
X= 5*cos(fi)*sin(psi); Y=5*sin(fi); Z= 5*cos(fi)*cos(psi);
```

Например, при psi = 20.0 и fi = 30.0 будет X= 1.48; Y= 2.5; Z=4.07.

- Каждое изображение сохраняется через клавишу prtsc в буферной памяти. Затем обрабатывается в графическом редакторе Paint — вырезается и сохраняется часть окна без заголовка, остаются только белый фон и треугольник. Сохраняются оба изображения как 24-разрядный BMP-файл — *triangle.bmp* и *triangle1.bmp*. При сохранении изображения через prtsc планируемое разрешение (500*500) сбивается — у меня получилось 619*628. Кадр с необходимым разрешением (500*500 или 300*300) устанавливается в редакторе Paint вручную.

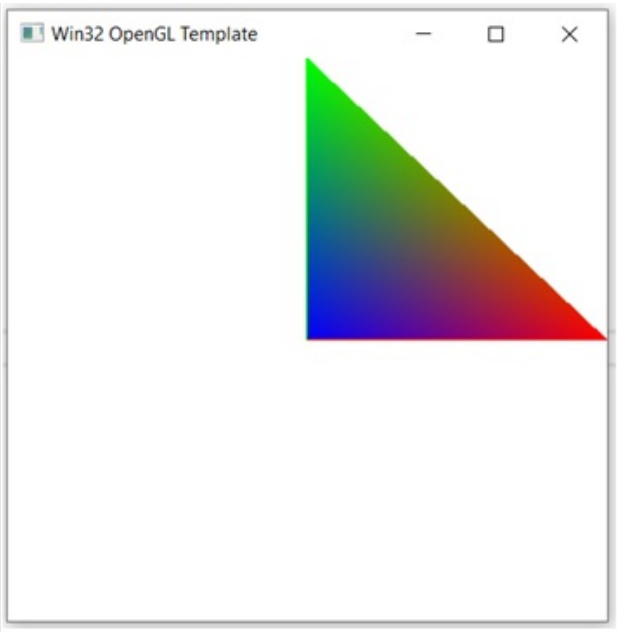


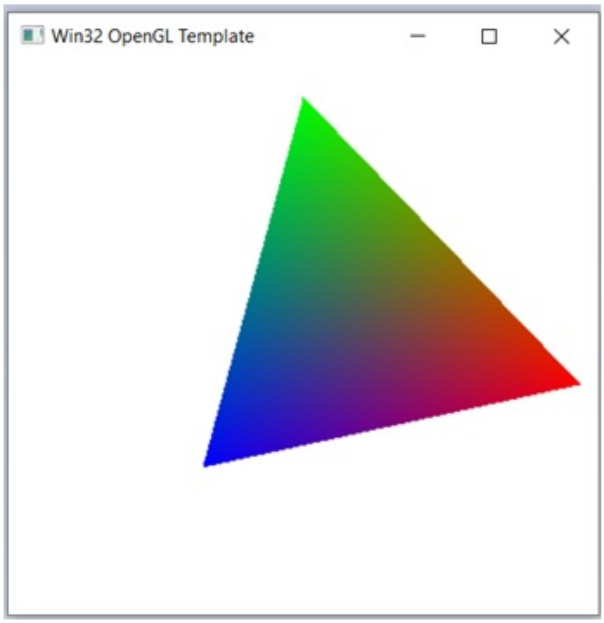
Обратите внимание, чтобы на главном изображении синяя вершина треугольника (начало координат) была по центру окна, а верхняя и правая вершины вершины треугольника находились на границах окна. На вспомогательном изображении определить начало координат сложнее. Вырезайте при редактировании все окно за исключением заголовочной части. Немного ориентируйтесь на то, какую часть Вы вырезали при создании предыдущего фото.

Подготовка программы для тестирования алгоритма

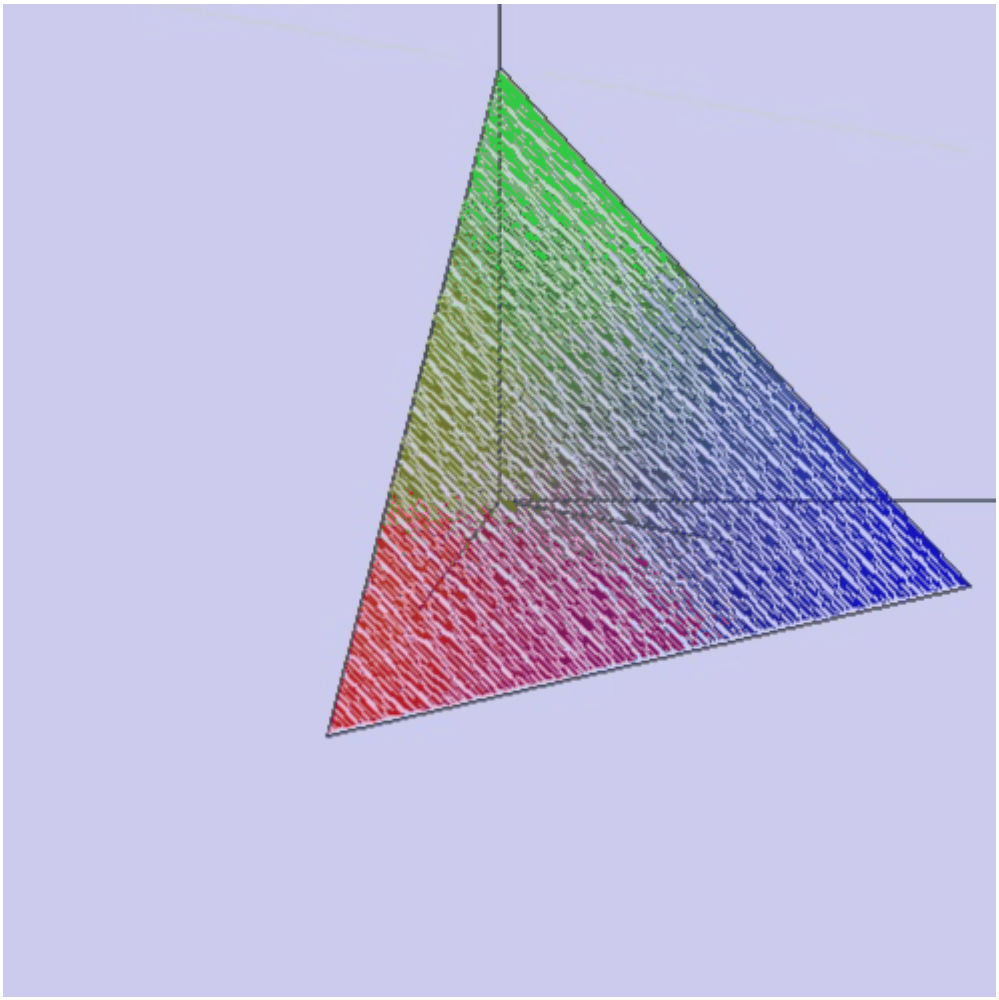
В статье 3D графика на основе WinApi C++ для выполнения контрольных заданий в качестве шаблона было использовано приложение, которое можно скачать с моего Googl-диска по ссылке 3d.zip. Модифицируем это приложение, чтобы оно позволяло протестировать алгоритм 3D-реконструкции по двум изображениям. Приложение должно обеспечивать:

- последовательную загрузку фотографий треугольника (*triangle.bmp* и *triangle1.bmp*).





- восстановление по этим фото 3D-облака точек, которые находятся в пределах периметра треугольника



Всю функциональность в основном связываем с последовательным нажатием на кнопки 1-3:

1. Считывание и формирование массива точек 1-го фото. Отображение в окне 1-го фото.
2. Считывание и формирование массива точек 2-го фото. Отображение в окне 2-го фото.

3. Определение координаты *Z* каждой точки и отображение облака точек.

> *Поворот треугольника мышкой уже обеспечен функциональностью приложения-шаблона.*

Прежде всего создайте папку *Data* в директории проекта приложения, где находятся исходные файла. В *Data* поместите файлы изображений *triangle.bmp* и *triangle1.bmp*.

Ниже приводятся описание модификации файлов приложения-шаблона.

main.cpp

Определение размеров окна в функции WndProc

```
// Создаем основное окно приложения
hWnd = CreateWindow(
(LPCWSTR)szClassName, // Имя класса
L"Шаблон WinAPI приложения", // Текст заголовка
WS_OVERLAPPEDWINDOW, // Стиль окна
50, 50, // Позиция левого верхнего угла
// К размерам фото еще добавляется 16 (по X) и 40 (по Y)
516, 540, // при разрешении фото 500* 500
//371, 365, // при разрешении фото 355*325
(HWND) NULL, // Указатель на родительское окно NULL
(HMENU) NULL, // Используется меню класса окна
(HINSTANCE)hInstance, // Указатель на текущее приложение
NULL );
```

Создание статических переменных и модификация события WM_CREATE:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT messg, WPARAM wParam, LPARAM lParam)
{
 PAINTSTRUCT ps;
 static RECT Rect;
 static HDC hdc, hCmpDC;
 static HBITMAP hBmp;
 static Action *action;
 static Engine *engine;
static BITMAP bm;
static bool r1 /*= false*/;
static bool r2 /*= false*/;
static bool r3 /*= false*/;
static bool r4 /*= false*/;
static bool w /*= false*/;
static int mx;
static int my;
static int mx1;
```

```
static int my1;
static int im;
static RGBQUAD **v1;
static RGBQUAD **v2;
switch (messg)
 {
 case WM_CREATE:
 engine = new Engine();
 action = new Action();
 engine->SetAction(action);
 // Флажки для управления отображением контента в окне
 r1 = true;
 r2 = true;
 r3 = true;
 r4 = true;
 im = 0;
break;
```

Модификация события WM_PAINT:

```
FillRect(hCmpDC, &Rect, brush);
 DeleteObject(brush);
// Загрузка картинок
 if (im == 1){ engine->LoadBMP(hWnd, hBmp, hCmpDC, bm, TEXT("Data/triangle.bmp"));}
 if (im == 2){engine->LoadBMP(hWnd, hBmp, hCmpDC, bm, TEXT("Data/triangle1.bmp"));}
 // Рисование
 //engine->Draw(hCmpDC);
 if (im == 3)
 {
 engine->Draw(hCmpDC);
 engine->ImagePixel(hCmpDC, v1, mx, my);
 }
 // Вывод на экран
 SetStretchBltMode(hdc, COLORONCOLOR);
```

Практически полностью изменяется событие WM_KEYDOWN

```
case WM_KEYDOWN:
 int KeyPressed;
 KeyPressed = int(wParam);
 if (KeyPressed == int('0')) {action->Transform_0();}
 if (KeyPressed == int('1'))
 {
  // Считывание и формирование массива точек 1-го фото (только 1 раз)
   if (r1 == true) {
     //int mx, my; // размеры фото
     v1 = engine->MatrixBMP("Data/triangle.bmp", mx, my); // выделяем память и читаем туда файл
     r1 = false;
   }
   // Отображение в окне 1-го фото
```

```
      im = 1;
    }
    if (KeyPressed == int('2'))
    {
     // Считывание и формирование массива точек 2-го фото (только 1 раз)
      if (r2 == true) {
        v2 = engine->MatrixBMP("Data/triangle1.bmp", mx1, my1); // выделяем память и читаем туда файл
        r2 = false;
      }
      // Отображение в окне 2-го фото
      im = 2;
    }
    if (KeyPressed == int('3'))
    {
     // определение соответствующих точек 2-х изображений и нахождение для каждой точки 3-й координат
    if (r3 == true) {
       action->Transform_4();
       engine->GetZ(v1, v2, mx, my); //Только один раз определяем координату Z для точек главного изо
        delete v2; // массив точек 2-го изображения можно удалить
        r3 = false;
       }
     im = 3;
    }
    InvalidateRect(hWnd, NULL, FALSE);
   break;
```

action.cpp

Замените определение функции Action::Transform_4()

```
void Action::Transform_4(){
  Matrix Tr;
  Tr.SetTranslationMatrix_4();
    CurrentMatrix.MultiplyMatrices(Tr);
    Tr.SetTranslationMatrix_5();
    CurrentMatrix.MultiplyMatrices(Tr);
}
```

matrix.cpp

Замените определение функций *SetRotationMatrixbySinCos*, *SetTranslationMatrix_4* и *SetTranslationMatrix_5* :

```
void Matrix::SetRotationMatrixbySinCos(double sinalpha, double cosalpha){
 SetUnit();
 //data[0][0] = cosalpha;
 //data[0][2] = sinalpha;
 //data[2][0] = -sinalpha;
```

```
 //data[2][2] = cosalpha;

 data[0][0] = cosalpha; // a    // a c p 0 //
 //data[0][1] = 1.0;    // c    // b d q 0 //
 data[0][2] = sinalpha; // p    // h f r 0//
 //data[0][3] = 1.0;    // 0    // m n l 1 //

 //data[1][0] = 0.1;    // b
 //data[1][1] = 0.0;    // d
 //data[1][2] = 0.1;    // q
 //data[1][3] = 0.1;    // 0

 data[2][0] = -sinalpha;  // h
 //data[2][1] = 0.25;     // f
 data[2][2] = cosalpha;   // r
 //data[2][3] = 1.0;      // 0

 //data[3][0] = 0.25;     // m
 //data[3][1] = 0.25;     // n
 //data[3][2] = 1.0;      // l
 //data[3][3] = 1.0;      // 1
}



void Matrix::SetTranslationMatrix_4(){
 SetUnit();
 data[0][0] = 0.94;  // a     // a c p 0 //
 //data[0][1] = 1.0; // c     // b d q 0 //
 data[0][2] = 0.342; // p     // h f r 0 //
 //data[0][3] = 1.0; // 0     // m n l 1 //

 //data[1][0] = 0.1;   // b
 //data[1][1] = 0.0;   // d
 //data[1][2] = -1.0;  // q
 //data[1][3] = 0.1;   // 0

 data[2][0] = -0.342;  // h       угол(Y) psi=+20
 //data[2][1] = 1.0;   // f
 data[2][2] = 0.94;    // r
 //data[2][3] = 0.0;   // 0

 //data[3][0] = 0.25;  // m
 //data[3][1] = 0.25;  // n
 //data[3][2] = 1.0;   // l
 //data[3][3] = 1.0;   // 1
}

void Matrix::SetTranslationMatrix_5(){
 SetUnit();
 //data[0][0] = 0.0; // a    // a c p 0 //
 //data[0][1] = 1.0; // c    // b d q 0 //
 //data[0][2] = 0.0; // p    // h f r 0 //
 //data[0][3] = 1.0; // 0    // m n l 1 //

 //data[1][0] = 0.1; // b
 data[1][1] = 0.866; // d
```

```
 data[1][2] = 0.5;   // q
 //data[1][3] = 0.1; // 0       угол (X) fi=-30

 //data[2][0] = 0.5;  // h
 data[2][1] = -0.5;   // f
 data[2][2] = 0.866;  // r
 //data[2][3] = 0.0;  // 0

 //data[3][0] = 0.25; // m
 //data[3][1] = 0.25; // n
 //data[3][2] = 1.0;  // l
 //data[3][3] = 1.0;  // 1
}
```

viewport.h

Поменяйте значение отступа (Margin) на 0

```
_Point T(_Point point);
 _Point T_inv(_Point point);
//void SetMargin(int _Margin = 10);
 void SetMargin(int _Margin = 0);
```

viewport.cpp

Добавьте по одной строке кода в определение функций:

```
/*
 Выполняет преобразование из координат [-1, 1] x [-1, 1]
 в координаты экрана, с учетом отступов (Margin)
*/
_Point Viewport::T(_Point point) {
 _Point TPoint;
 TPoint.x = Margin + (1.0 / 2) * (point.x + 1) * (Width - 2 * Margin);
 TPoint.y = Margin + (-1.0 / 2) * (point.y - 1) * (Height - 2 * Margin);
 TPoint.z = Margin + (1.0 / 2) * (point.z + 1) * (Width - 2 * Margin);
 return TPoint;
}
/*
 Выполняет преобразование из координат экрана, в координаты
 [-1, 1] x [-1, 1] с учетом отступов (Margin)
*/
_Point Viewport::T_inv(_Point point) {
 _Point TPoint;
 TPoint.x = double(point.x - Margin) / (1.0 / 2) / (Width - 2 * Margin) - 1;
 TPoint.y = double(point.y - Margin) / (-1.0 / 2) / (Height - 2 * Margin) + 1;
 TPoint.z = double(point.z - Margin) / (1.0 / 2) / (Width - 2 * Margin) - 1;
```

```
 return TPoint;
}
```

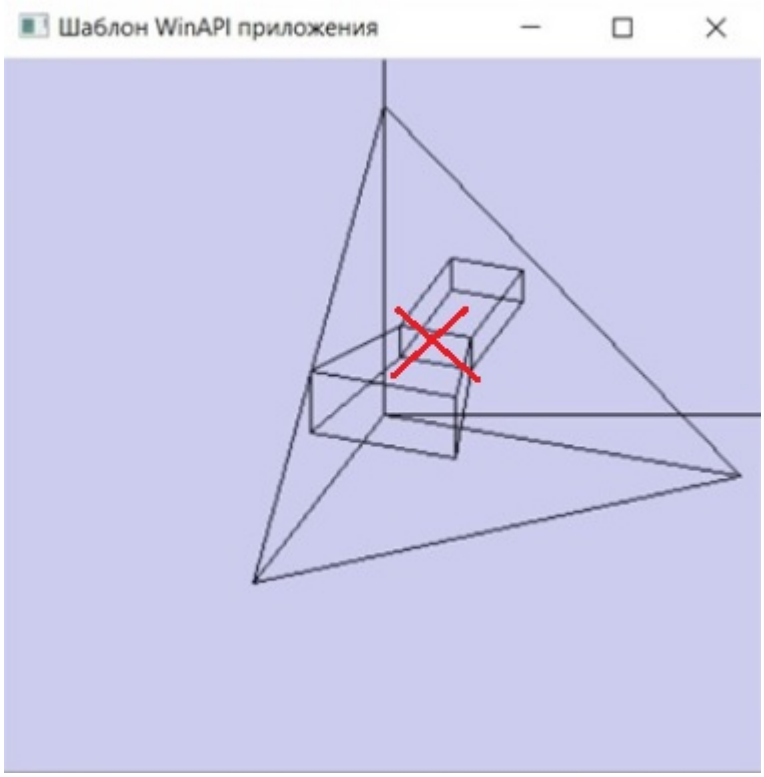
engine.h

В класс Engine добавляются функции и переменные для загрузки фото в окно, считывания информации из BMP-файла и определения координаты Z точек.

```
#include <windows.h>
#include "action.h"
#include "viewport.h"
#include <stdio.h>
class Engine{
 //Matrix current_rot;
 Action *action;
 unsigned short read_u16(FILE *fp);
 unsigned int read_u32(FILE *fp);
 int read_s32(FILE *fp);
public:
 Viewport viewport;
 void Draw(HDC hdc);
 void SetAction(Action *_action);
 int LoadBMP(HWND hWnd, HBITMAP hBmp, HDC hCmpDC, BITMAP bm, LPCWSTR fileName);
 RGBQUAD ** MatrixBMP(const char *fname, int &mx, int &my);
 void GetZ(RGBQUAD **rgb, RGBQUAD **rgb1, int mx, int my);
 void ImagePixel(HDC hdc, RGBQUAD **rgb,int mx, int my);
};
```

engine.cpp

В этом файле определяются функции класса Engine. В функции Draw место рупорной антенны создается каркас треугольника.



В файле *engine.cpp* очень много изменений. Поэтому замените текст файла полностью. Особое внимание обратите на текст, выделенный красным цветом — он имеет непосредственное отношение к алгоритму 3D-реконструкции.

```
//#include <windows.h>
#include "geometry.h"
#include "matrix.h"
#include "engine.h"
/*
 Привязываем к движку объект, который отвечает за действия пользователя
*/
void Engine::SetAction(Action *_action)
{
 action = _action;
}
/*
 Выводит графику на контекст hdc
*/
void Engine::Draw(HDC hdc){
 _Point p[12];
 p[1].x = 1.0;
 p[1].y = 0.0;
 p[1].z = 0.0;
 p[2].x = 0.0;
 p[2].y = 1.0;
 p[2].z = 0.0; // 0.5 = 1
 p[3].x = 0.0;
 p[3].y = 0.0;
 p[3].z = 1.0;
 // локальная система координат
 p[4].x = 0.0;
 p[4].y = 0.0;
```

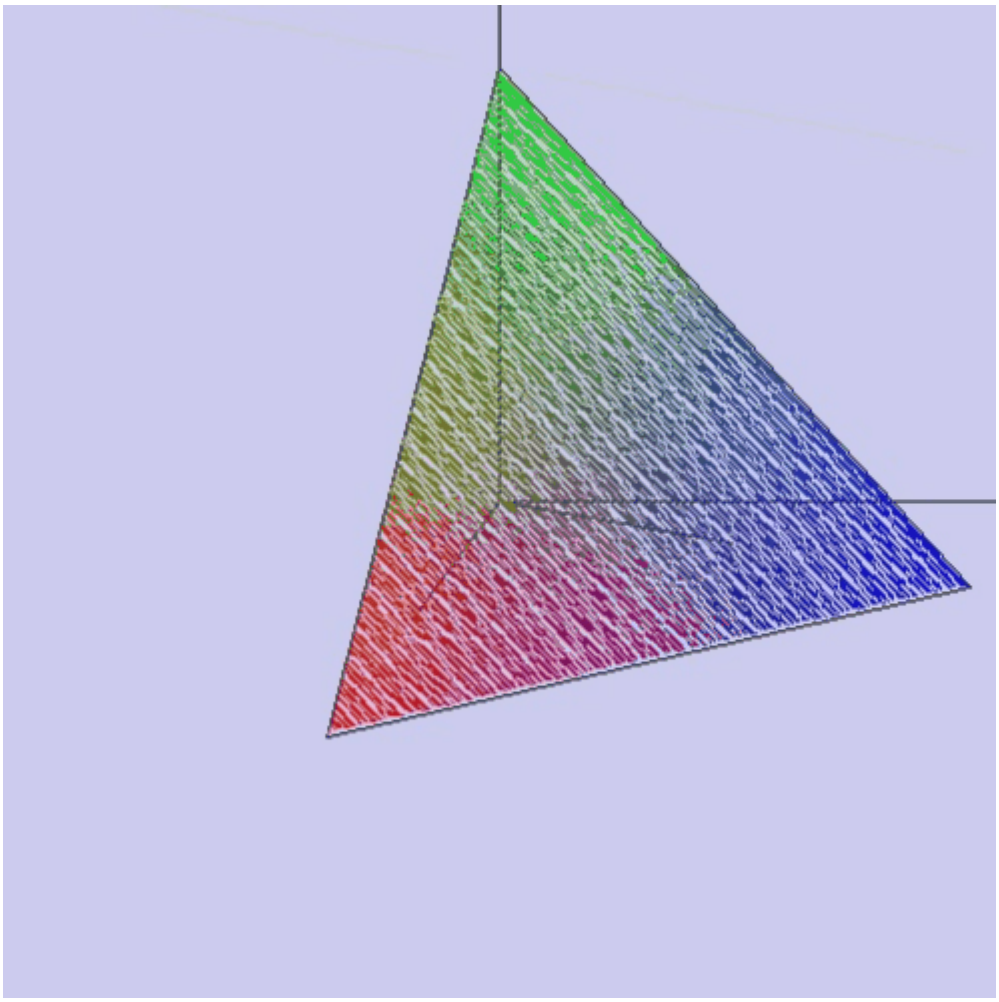
```
 p[4].z = 0.0;
 p[5].x = 0.5;
 p[5].y = 0.0;
 p[5].z = 0.0;
 p[6].x = 0.0;
 p[6].y = 0.5;
 p[6].z = 0.0;
 p[7].x = 0.0;
 p[7].y = 0.0;
 p[7].z = 0.5;
for (int i = 1; i <8; i++){
 //Вращение и перемещение осуществляется в логических координатах
 action->CurrentMatrix.ApplyMatrixtoPoint(p[i]);
 // Переход из логических в оконные координаты.
 p[i] = viewport.T(p[i]);
 }
// треугольник
 MoveToEx(hdc, p[1].x, p[1].y, NULL);
 LineTo(hdc, p[2].x, p[2].y);
 LineTo(hdc, p[3].x, p[3].y);
 LineTo(hdc, p[1].x, p[1].y);
// оси координат
int i = 4;
for (int j = 1; j <= 3; j++){
MoveToEx(hdc, p[i].x, p[i].y, NULL);
LineTo(hdc, p[i+j].x, p[i+j].y);
}
// глобальная система координат
p[8].x = 0.0;
p[8].y = 0.0;
p[8].z = 0.0;
p[9].x = 1.0;
p[9].y = 0.0;
p[9].z = 0.0;
p[10].x = 0.0;
p[10].y = 1.0;
p[10].z = 0.0;
p[11].x = 0.0;
p[11].y = 0.0;
p[11].z = 1.0;
for (int i = 1; i <12; i++){
 // Переход из логических в оконные координаты.
 p[i] = viewport.T(p[i]);
}
// оси координат
i = 8;
for (int j = 1; j <= 3; j++){
 MoveToEx(hdc, p[i].x, p[i].y, NULL);
 LineTo(hdc, p[i+j].x, p[i+j].y);
}
}
int Engine::LoadBMP(HWND hWnd, HBITMAP hBmp, HDC hCmpDC, BITMAP bm, LPCWSTR fileName){
 // Загрузка картинки
hBmp = (HBITMAP)LoadImage(NULL, fileName, IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE | LR_CREATEDIBSECTIC
if (hBmp == NULL)
 {
 MessageBoxW(hWnd, TEXT("Файл не найден"), TEXT("Загрузка изображения"), MB_OK | MB_ICONHAND);
```

```
 DestroyWindow(hWnd);
 return 1;
 }
GetObject(hBmp, sizeof(bm), &bm);
 SelectObject(hCmpDC, hBmp);
 ReleaseDC(hWnd, hCmpDC);
 return 1;
}

void Engine::GetZ(RGBQUAD **rgb, RGBQUAD **rgb1, int mx, int my){
 _Point p;
 _Point p1;
double dz=0.01, pxy;
 int i, j, i1, j1, SUM, dmin;
dmin = 1;
 int dR, dG, dB;
for (j = 1; j < my; j++) {
 for (i = 1; i < mx; i++) {
 SUM = rgb[i][j].rgbRed + rgb[i][j].rgbGreen + rgb[i][j].rgbBlue;
 if (SUM < 765) { // Отфильтровываем светлые точки (255,255,255)
 // переходим от координат изображения (снизу вверх) к оконным координатам (сверху вниз)
 p.x = i;
 p.y = my - j;
 rgb[i][j].rgbReserved = 127; // z=0 rgb[i][j].rgbReserved используется для хранения координаты Z
 p.z = mx *rgb[i][j].rgbReserved / 255;
 //Выполняется преобразование оконных координат в логические координаты x[-1, 1] y[-1, 1] z[-1, 1]
 p = viewport.T_inv(p);
 if (p.x*p.x + p.y*p.y + p.z*p.z < 1.0) // Отфильтровываем точки за пределами сферы единичного рад
 do {
 p.z = p.z + dz;
 if (p.x*p.x + p.y*p.y + p.z*p.z >= 1.0){ p.z = 1.1; break; }; // Отфильтровываем точки за предела
 p1.x = p.x *0.94 - p.z*0.342;  //sin20=0.342 cos20=0.94
 p1.y = p.y *0.866 - p.z*0.47 - p.x*0.171; // cos30=0.866 sin30=0.5
 //p1.z = 0.0; в данном случае координата z не имеет значения
 p1 = viewport.T(p1); // Переход из логических в оконные координаты.
 // Переход от оконных координат к пикселам изображения (снизу вверх)
 i1 = p1.x;
 j1 = my - p1.y;
 dR = abs(rgb1[i1][j1].rgbRed - rgb[i][j].rgbRed);
 dG = abs(rgb1[i1][j1].rgbGreen - rgb[i][j].rgbGreen);
 dB = abs(rgb1[i1][j1].rgbBlue - rgb[i][j].rgbBlue);

} while ((dR > dmin || dG > dmin || dB > dmin) && p.z <= 1.0);
 if (p.z <= 1) {
 p = viewport.T(p); // Переход из логических в оконные координаты (главное изображение)
 rgb[i][j].rgbReserved = p.z * 255 / mx; // запоминаем координату z в байтах (0-255)
 }
 else
 {
 rgb[i][j].rgbRed = 255; rgb[i][j].rgbGreen = 255; rgb[i][j].rgbBlue = 255;
 };
 }
}
}
}
void Engine::ImagePixel(HDC hdc, RGBQUAD **rgb, int mx, int my) {
COLORREF clr;
```

```
 _Point p;
 int i, j, SUM;
 //int dR, dG, dB;
 //выводим результат
 for (j = 0; j < my; j++) {
 for (i = 0; i < mx; i++) {
 SUM = rgb[i][j].rgbRed + rgb[i][j].rgbGreen + rgb[i][j].rgbBlue;
 if (SUM < 755) { // Отфильтровываем светлые точки (255,255,255)
 p.x = i;
 p.y = my - j;

//rgb[i][j].rgbReserved = 127; // z=0
 p.z = mx *rgb[i][j].rgbReserved / 255;
 //if (p.x == mx / 2 && p.y == mx / 2) { p.z = mx; }; // рисуется точка
 //Выполняется преобразование из координат экрана в логические координаты x[-1, 1] y[-1, 1] z[-1,
 p = viewport.T_inv(p);
//
 //p.z = 0.5;
 // Блок выполняется при каждой перерисовке
 //Вращение и перемещение осуществляется в логических координатах
 action->CurrentMatrix.ApplyMatrixtoPoint(p);
 // Переход из логических в оконные координаты.
 p = viewport.T(p);
 // Рисуем точку
 clr = RGB(rgb[i][j].rgbRed, rgb[i][j].rgbGreen, rgb[i][j].rgbBlue);
 SetPixel(hdc, p.x, p.y, clr);
 }
 }
 }
}
RGBQUAD ** Engine::MatrixBMP(const char *fname, int &mx, int &my)
{
 FILE * pFile = fopen(fname, "rb");
// считываем заголовок файла
 BITMAPFILEHEADER header;
 header.bfType = read_u16(pFile);
 header.bfSize = read_u32(pFile);
 header.bfReserved1 = read_u16(pFile);
 header.bfReserved2 = read_u16(pFile);
 header.bfOffBits = read_u32(pFile);
// считываем заголовок изображения
 BITMAPINFOHEADER bmiHeader;
 bmiHeader.biSize = read_u32(pFile);
 bmiHeader.biWidth = read_s32(pFile);
 bmiHeader.biHeight = read_s32(pFile);
 bmiHeader.biPlanes = read_u16(pFile);
 bmiHeader.biBitCount = read_u16(pFile);
 bmiHeader.biCompression = read_u32(pFile);
 bmiHeader.biSizeImage = read_u32(pFile);
 bmiHeader.biXPelsPerMeter = read_s32(pFile);
 bmiHeader.biYPelsPerMeter = read_s32(pFile);
 bmiHeader.biClrUsed = read_u32(pFile);
 bmiHeader.biClrImportant = read_u32(pFile);
 RGBQUAD **rgb = new RGBQUAD*[bmiHeader.biWidth];
 for (int i = 0; i < bmiHeader.biWidth; i++) {
 rgb[i] = new RGBQUAD[bmiHeader.biHeight];
 }
```

```
int kr = (int)bmiHeader.biWidth * 3 % 4;
 if (kr != 0) {kr = 4 - kr; }
for (int j = 0; j < bmiHeader.biHeight; j++) {
 for (int i = 0; i < bmiHeader.biWidth; i++) {
 rgb[i][j].rgbBlue = getc(pFile);
 rgb[i][j].rgbGreen = getc(pFile);
 rgb[i][j].rgbRed = getc(pFile);
 }
// пропускаем последние 1-3 байта в конце строки для обеспечения кратности 4
 for (int i = 0; i < kr; i++) {
 getc(pFile);
 }
}
// Передаем значения ширины и высоты глобальным переменным
 mx = bmiHeader.biWidth;
 my = bmiHeader.biHeight;
fclose(pFile);
 return rgb;
}
unsigned short Engine::read_u16(FILE *fp)
{
 unsigned char b0, b1;
 b0 = getc(fp);
 b1 = getc(fp);
return ((b1 << 8) | b0);
}
unsigned int Engine::read_u32(FILE *fp)
{
 unsigned char b0, b1, b2, b3;
 b0 = getc(fp);
 b1 = getc(fp);
 b2 = getc(fp);
 b3 = getc(fp);
return ((((((b3 << 8) | b2) << 8) | b1) << 8) | b0);
}
int Engine::read_s32(FILE *fp)
{
 unsigned char b0, b1, b2, b3;
 b0 = getc(fp);
 b1 = getc(fp);
 b2 = getc(fp);
 b3 = getc(fp);
 return ((int)(((((b3 << 8) | b2) << 8) | b1) << 8) | b0);
}
```

Результат запуска приложения с нажатием на клавиши 1-3 а затем вращения треугольника курсором от мышки.



При возникновении ошибки типа error C4996: 'fopen': This function or variable may be unsafe в свойствах проекта отключите предупреждения, генерируемые _CRT_SECURE_NO_DEPRECATE. Как это сделать, смотри здесь.

В приложении осталось много кода, который не используется при решении поставленной задачи. Ознакомившись с программой можете самостоятельно его удалить.

Тестирование алгоритма с реальным объектом

Тест №1

В случае, когда фотографии получены с обычного фотоаппарата, взаимное положение камер может быть определено только лишь приближенно — с погрешностью углов в пределах 5 градусов.

Ниже приводится результат (рис. 10 -12) использования программы 3D реконструкции детской игрушки по 2 фотографиям. Фотографии были получены под углами поворота камеры: psi = 20.0 и fi = 30.0. Углы определялись с погрешностью в пределах 5 градусов.

Тестовые фото игрушки «Знайка» можно скачать здесь Data.

Рис. 10. Фото 1

Рис. 11. Фото 2

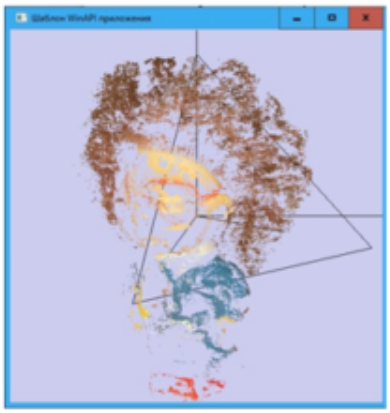
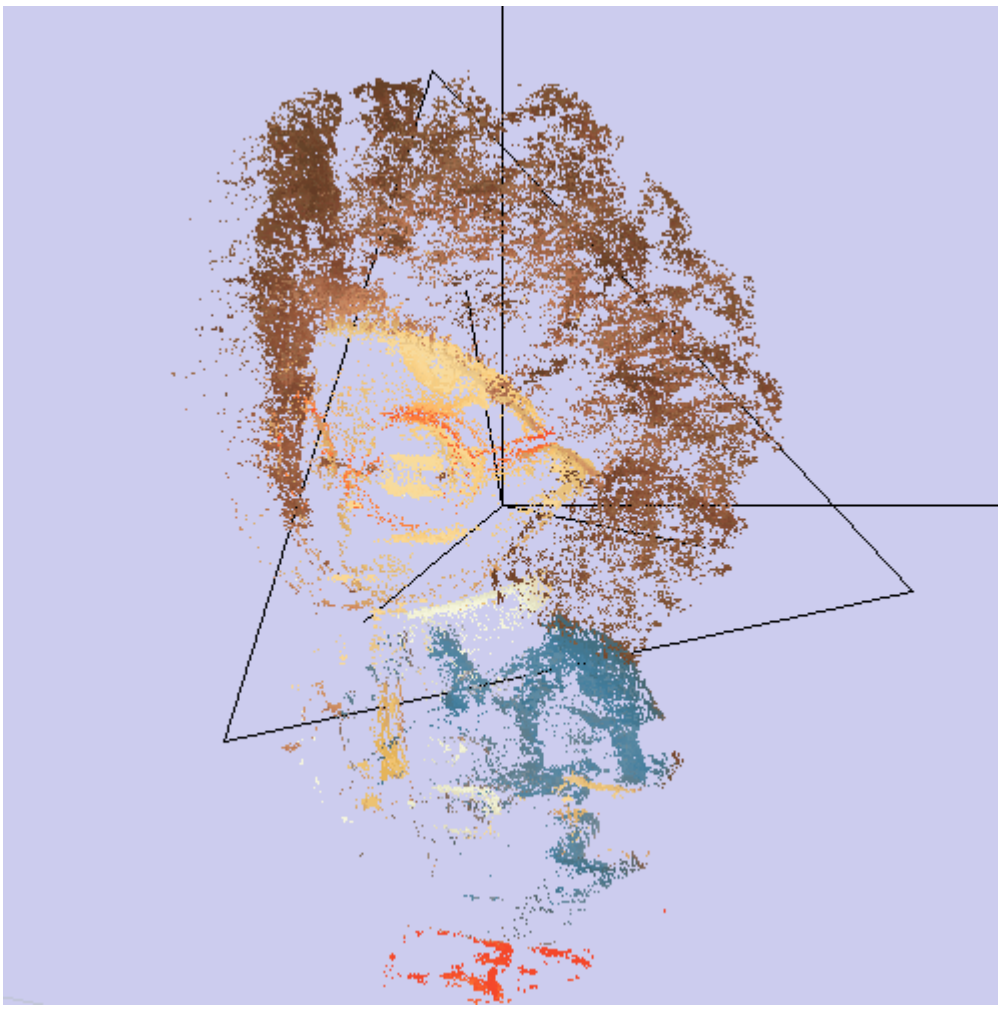


Рис. 12. Облако 3D точек



Полученный результат свидетельствует о том, что даже при такой невысокой точности определения положения камеры был получен относительно неплохой результат.

Тест №2

Для дальнейшего тестирования задачи было разработано приложение под мобильное устройство с операционной системой Android, которая загружает фотографии и по ним реконструирует 3D объект в виде облака точек. Результаты приведены (рис. 13 — 14).

Рис. 13. Тест № 1 приложения под Android device

Рис. 14. Тест № 2 приложения под Android device

Программная реализация Android приложения на этом сайте не приводится. Есть только лишь заготовка для такого приложения (см. здесь). Можете попробовать самостоятельно реализовать такое приложение.

Контрольные задания:

1. Подготовить программы по описанию (см. Создание фото… и ….тестирования алгоритма 3D-реконструкции). Есть альтернативный и более простой способ — скачать проекты программ с моего Google диска (3dgl-ph.zip и 3d-r.zip)
2. Протестировать алгоритм на основе фото треугольников для приводимого в описании примера. Напомню, фото были получены под углами *psi=20* и *fi=30*.
3. Протестировать алгоритм на основе фото игрушки «Знайка». Эти фото также были получены под углами psi=20 и fi=30.
4. Подготовить фото треугольников в соответствии с вариантом (значение углов *psi* и *fi* взять равными углам *alpha* и *beta* из таблицы задания 2).
5. Самостоятельно подготовить фото какого-либо реального объекта и протестировать алгоритм.

Выводы

1. В работе описан и программно реализован (на C++) алгоритм 3D-реконструкции по 2-м изображениям. Он построен на основе решения одной из множества задач аффинных преобразований в однородных координатах. Алгоритм описан только для простейшего случая ортогонального проецирования, когда взаимное положение камер определяется только 2-мя угловыми параметрами.
2. Алгоритм может быть обобщен для случая, когда взаимное положение камер определяется большим количеством параметров, включая аппарат центрального проецирования и учитывая не только вращение, но и перемещение устройства в пространстве. Реализация такого алгоритма будет возможна, когда мобильные устройства смогут с достаточной точностью определять параметры перемещения и расстояния в пространстве. Недостаток параметров также может быть компенсирован за счет совершенствования алгоритма, например – путем

нахождения на обоих фото особых точек и установления соответствия между ними на изображениях.

3. Даже аппарат центрального проецирования есть всего лишь идеализированной моделью процесса формирования изображения в цифровой камере. Для того, чтобы максимально приблизить модель к реальному процессу, необходимо решить ряд задач предварительной обработки изображений, отделения фона, учесть потери качества, связанные с сжатием изображений, т.е., – решить комплекс задач компьютерного зрения.
4. Программа, в которой реализован алгоритм, восстанавливает по фотографиям лишь облако точек пространственного объекта. Однако, она может быть легко модифицирована под поверхностную модель, например, методом триангуляции Делоне. Триангуляция – это разбиение геометрического объекта на треугольники, которые строятся по ближним точкам. При этом цвет треугольника интерполируется по цвету этих точек. В качестве примера использования триангуляции см. раздел Моделирование ландшафта из статьи Имитация полета крылатой ракеты на OpenGL WinApi C++.

Полезные ссылки:

- Stereo Vision: Depth Estimation between object and camera
- 3D Reconstruction from Videos: LASR
- 3D-сканирование структурированного света
- Использование открытых исходных кодов для разработки 3D сканера и соответствующего программного обеспечения для 3D реконструкции моделей.
- Parallax Images
- Восстановление трехмерных моделей активным параллаксным методом
- Introduction to Epipolar Geometry and Stereo Vision
- learnopencv/EpipolarGeometryAndStereoVision/github
- Automated virtual tour
- 3д-сканирование: Фотограмметрия
- 3D Reconstruction with Stereo Images -Part 1: Camera Calibration
- 3D-реконструкция по изображениям — Станислав Протасов
- 3D-сканирование с помощью фотокамеры — Виталий Окулов
- 123D Catch получение 3D модели из фотографий !!!!
- Обучение по использованию 3д-сканера
- Говорящие руки. Технология 3D-сканирования !!!!
- Топ-10 приложений для 3D-принтеров
- Faceworx + MeshMixer = 3d модель по фото !!!!

Автор: Николай Свирневский

Раздел: Computer vision Geometric modeling