Компьютерная графика

лекция 8

Структуры

```
struct vertex3 { GLdouble x, y, z; };
// Структура для вершины

struct color3 { GLdouble r, g, b; };
// Структура для цвета

struct tvertex2 { GLfloat x, y; };
// Структура для текстурной координаты
```

Структуры

```
struct line2 { unsigned int p1, p2; };
 // Структура для линии

struct triangle3 { unsigned int p1, p2, p3; };
// Структура для треугольной грани

struct quad4 { unsigned int p1, p2, p3, p4; };
// Структура для четырехугольной грани
```

Функции для структур

```
// создание структуры вектора
vertex3 vertex(GLdouble x, GLdouble y, GLdouble z) {
    vertex3 result = { x, y, z };
    return result;
}

// создание структуры цвета
color3 color(GLdouble r, GLdouble g, GLdouble b) {
    color3 result = { r, g, b };
    return result;
}
```

Функции для структур

```
// создание структуры текстурной координаты
tvertex2 tvertex(float x, float y)
{
    tvertex2 tv = { x, y };
    return tv;
}
// создание структуры линии
line2 line(int p1, int p2)
{
    line2 l;
    l.p1 = p1; l.p2 = p2;
    return l;
}
```

Функции для структур

```
// создание структуры треугольника
triangle3 triangle(int p1, int p2, int p3)
{
    triangle3 t;
    t.p1 = p1; t.p2 = p2; t.p3 = p3;
    return t;
}
// создание структуры четырехугольника
quad4 quad(int p1, int p2, int p3, int p4)
{
    quad4 q;
    q.p1 = p1; q.p2 = p2; q.p3 = p3; q.p4 = p4;
    return q;
}
```

Класс Item

```
class Item { // базовый класс
protected:
    vertex3 pos;    vertex3 angle;     color3 color;        float size;
public:
    Item();
    virtual void Draw(); // рисование
    virtual void Init();  // инициализация
    void SetSize(float size);  // установка размера
    void SetPos(vertex3 pos); // установка позиции
    void SetAngle(vertex3 angle); //  установка угла
    void SetColor(color3 color); // установка цвета
    void Rotate(); // применение поворота
};
```

Класс Item

```
Item::Item()
{
    pos.x = 0; pos.y = 0; pos.z = 0;
    angle.x = 0; angle.y = 0; angle.z = 0;
    color.r = 1; color.g = 1; color.b = 1;
    size = 1;
};
void Item::Draw(){}
void Item::Init(){}
void Item::SetSize(float size)
{
    this->size = size;
}
```

Класс Item

```
void Item::SetPos(vertex3 pos)
{
    this->pos = pos;
}
void Item::SetAngle(vertex3 angle)
{
    this->angle = angle;
}
void Item::SetColor(color3 color)
{
    this->color = color;
}
```

Класс Item

```
void Item::Rotate()
{
    glTranslated(pos.x, pos.y, pos.z);
    glRotated(angle.x, 1, 0, 0);
    glRotated(angle.y, 0, 1, 0);
    glRotated(angle.z, 0, 0, 1);
    glTranslated(-pos.x, -pos.y, -pos.z);
}
```

Класс Point

```
class Point : public Item
{
public:
    Point(vertex3 pos, color3 color, float size);
    virtual void Draw();
};

Point::Point(vertex3 pos, color3 color, float size):Item()
{
    SetPos(pos);  SetColor(color);  SetSize(size);
}
```

Класс Point

```
void Point::Draw()
{
    glPointSize(size);
    glColor3d(color.r, color.g, color.b);
    glPushMatrix();
    glTranslated(pos.x, pos.y, pos.z);
    glBegin(GL_POINTS);
     glVertex3d(0, 0, 0);
    glEnd();
    glPopMatrix();
}
```

Класс Line

```
class Line : public Point {
protected:
    vertex3 pos2;
    color3 color2;

public:
   // конструктор с координатами, цветом и размером
    Line(vertex3 pos1, vertex3 pos2, color3 color, float size);
// конструктор с координатами, двумя цветами и размером
    Line(vertex3 pos1, vertex3 pos2, color3 color, color3 color2,
     float size);
    virtual void Draw();
};
```

Класс Line

```
Line::Line(vertex3 pos1, vertex3 pos2, color3 color, float size)
:Point(pos1, color, size)
{
    this->pos2 = pos2;
    color2 = color;
}
Line::Line(vertex3 pos1, vertex3 pos2, color3 color, color3
    color2, float size) : Point(pos1, color, size)
{
    this->pos2 = pos2;
    this->color2 = color2;
}
```

Класс Line

```
void Line::Draw()
{
    glPushMatrix();
    Rotate();
    glLineWidth(size);
    glBegin(GL_LINES);
      glColor3d(color.r, color.g, color.b);
      glVertex3d(pos.x, pos.y, pos.z);
      glColor3d(color2.r, color2.g, color2.b);
      glVertex3d(pos2.x, pos2.y, pos2.z);
    glEnd();
    glPopMatrix();
}
```

Класс Triangle

```
class Triangle : public Line
{
protected:
    vertex3 pos3;
    color3 color_3;
public:
 // конструктор с координатами, цветом и размером
    Triangle(vertex3 pos1, vertex3 pos2, vertex3 pos3, color3
    color);
 // конструктор с координатами, тремя цветами и размером
    Triangle(vertex3 pos1, vertex3 pos2, vertex3 pos3, color3
     color, color3 color2, color3 color_3);
    virtual void Draw();
};
```

Класс Triangle

```
Triangle::Triangle(vertex3 pos1, vertex3 pos2, vertex3 pos3,
    color3 color) :Line(pos1, pos2, color, size)
{
    this->pos3 = pos3;
    color_3 = color;
}

Triangle::Triangle(vertex3 pos1, vertex3 pos2, vertex3 pos3,
    color3 color, color3 color2, color3 color_3) :Line(pos1, pos2,
    color, color2, size)
{
    this->pos3 = pos3;    this->color_3 = color_3;
}
```

Класс Triangle

```
void Triangle::Draw()
{   glPushMatrix();
    Rotate();
    glBegin(GL_TRIANGLES);
        glColor3d(color.r, color.g, color.b);
        glVertex3d(pos.x, pos.y, pos.z);
        glColor3d(color2.r, color2.g, color2.b);
        glVertex3d(pos2.x, pos2.y, pos2.z);
        glColor3d(color_3.r, color_3.g, color_3.b);
        glVertex3d(pos3.x, pos3.y, pos3.z);
    glEnd();
    glPopMatrix();
}
```

Класс Quad

```
class Quad : public Triangle
{
protected:
    vertex3 pos4; color3 color4;
public:
 // конструктор с координатами, цветом и размером
    Quad(vertex3 pos1, vertex3 pos2, vertex3 pos3, vertex3
    pos4, color3 color);
 // конструктор с координатами, 4мя цветами и размером
    Quad(vertex3 pos1, vertex3 pos2, vertex3 pos3, vertex3
    pos4, color3 color, color3 color2, color3 color_3, color3
    color4);
    virtual void Draw();
};
```

Класс Quad

```
Quad::Quad(vertex3 pos1, vertex3 pos2, vertex3 pos3, vertex3
    pos4, color3 color) :Triangle(pos1, pos2, pos3, color)
{
    this->pos4 = pos4;
    color4 = color;
}

Quad::Quad(vertex3 pos1, vertex3 pos2, vertex3 pos3, vertex3
    pos4, color3 color, color3 color2, color3 color_3, color3
    color4) :Triangle(pos1, pos2, pos3, color, color2, color_3)
{
    this->pos4 = pos4;
    this->color4 = color4;
}
```

Класс Quad

```
void Quad::Draw()
{   glPushMatrix();    Rotate();
    glBegin(GL_QUADS);
        glColor3d(color.r, color.g, color.b);
        glVertex3d(pos.x, pos.y, pos.z);
        glColor3d(color2.r, color2.g, color2.b);
        glVertex3d(pos2.x, pos2.y, pos2.z);
        glColor3d(color_3.r, color_3.g, color_3.b);
        glVertex3d(pos3.x, pos3.y, pos3.z);
        glColor3d(color4.r, color4.g, color4.b);
        glVertex3d(pos4.x, pos4.y, pos4.z);
    glEnd();
    glPopMatrix();
}
```

Класс QuadT

```
class QuadT : public Quad
{
protected:
    tvertex2 tv1, tv2, tv3, tv4;
    string name_texture;
public:
    QuadT(vertex3 pos1, vertex3 pos2, vertex3 pos3, vertex3
    pos4, color3 color, string name_texture);
    QuadT(vertex3 pos1, vertex3 pos2, vertex3 pos3, vertex3
    pos4, color3 color, string name_texture, tvertex2 tv1, tvertex2
    tv2, tvertex2 tv3,  tvertex2 tv4);
    virtual void Draw();
};
```

Класс QuadT

```
QuadT::QuadT(vertex3 pos1, vertex3 pos2, vertex3 pos3, vertex3
    pos4, color3 color, string name_texture) :Quad(pos1, pos2, pos3,
    pos4, color)
{
    this->name_texture = name_texture;
    this->tv1 = tvertex(0, 0);     this->tv2 = tvertex(0, 1);
    this->tv3 = tvertex(1, 1);     this->tv4 = tvertex(1, 0);
}
QuadT::QuadT(vertex3 pos1, vertex3 pos2, vertex3 pos3, vertex3
    pos4, color3 color, string name_texture, tvertex2 tv1, tvertex2 tv2,
    tvertex2 tv3, tvertex2 tv4) :Quad(pos1, pos2, pos3, pos4, color)
{
    this->name_texture = name_texture;
    this->tv1 = tv1; this->tv2 = tv2;
    this->tv3 = tv3; this->tv4 = tv4;
}
```

Класс QuadT

```
void QuadT::Draw()
{
    glPushMatrix();
    Rotate();
    int tindex = SCENE::getTextureIndex(name_texture);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, tindex);
    glBegin(GL_QUADS);
     glTexCoord2f(tv1.x, tv1.y);
     glColor3d(color.r, color.g, color.b);
     glVertex3d(pos.x, pos.y, pos.z);
```

Класс QuadT

```
    glTexCoord2f(tv2.x, tv2.y);
    glColor3d(color2.r, color2.g, color2.b);
    glVertex3d(pos2.x, pos2.y, pos2.z);
    glTexCoord2f(tv3.x, tv3.y);
    glColor3d(color_3.r, color_3.g, color_3.b);
    glVertex3d(pos3.x, pos3.y, pos3.z);
    glTexCoord2f(tv4.x, tv4.y);
    glColor3d(color4.r, color4.g, color4.b);
    glVertex3d(pos4.x, pos4.y, pos4.z);
  glEnd();
  glPopMatrix();
  glDisable(GL_TEXTURE_2D);
}
```

Класс Item3D

```
class Item3D : public Item
{
protected:
    int tindex;
    int drawtype;
    vector <vertex3> points;
    vector <line2> lines;
    vector <triangle3> triangles;
    vector <quad4> quads;
public:
    Item3D();
    ~Item3D();
    virtual void Draw();
};
```

Класс Item3D

```
Item3D::Item3D():Item()
{
    tindex = -1;
    drawtype = 0;
};

Item3D::~Item3D()
{
    points.clear();
    lines.clear();
    quads.clear();
    triangles.clear();
}
```

Класс Item3D

```
void Item3D::Draw()
{
    glPushMatrix();
    Rotate();
    if ((drawtype==0)) // рисование только точками
          for (unsigned i = 0; i<points.size(); i++)
               DrawPoint(points[i], 10, color);

    if ((drawtype==1)) // рисование только линиями
     for (unsigned i = 0; i<lines.size(); i++)
          DrawLine(points[lines[i].p1], points[lines[i].p2], color, 2);
```

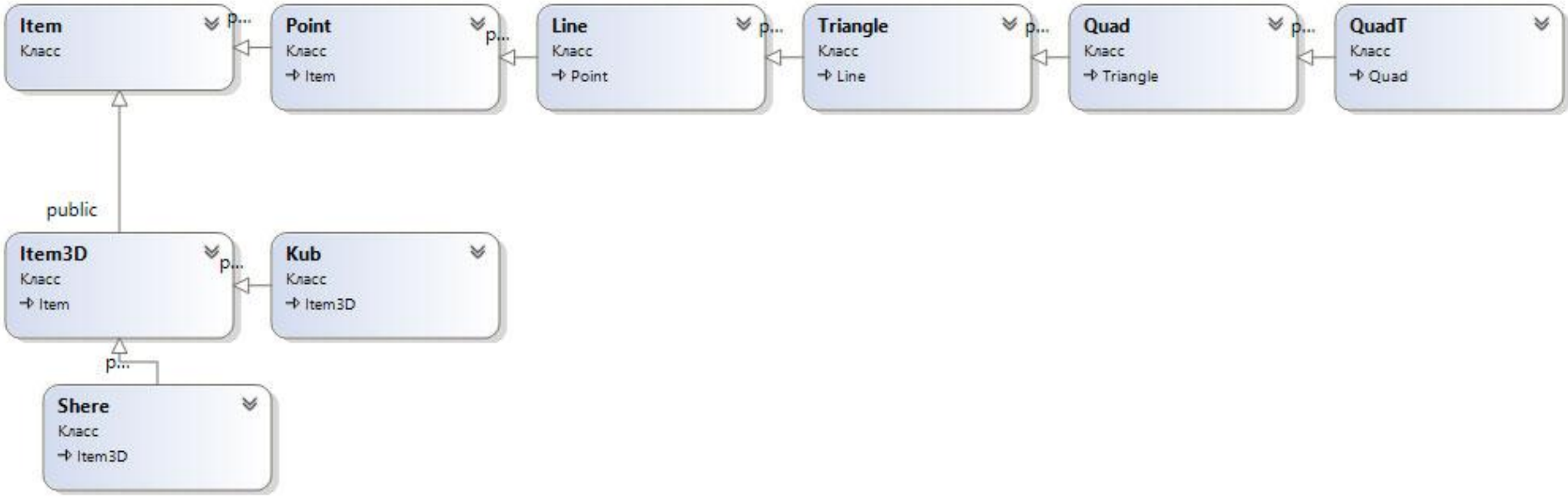
Класс Item3D

```
if ((drawtype==2)) // рисование треугольниками и
                   четырехугольниками
 {
     for (unsigned i = 0; i<triangles.size(); i++)
         DrawTriangle(points[triangles[i].p1],

                 points[triangles[i].p2],
                 points[triangles[i].p3], color);
     for (unsigned i = 0; i<quads.size(); i++)
         DrawQuad(points[quads[i].p1], points[quads[i].p2],
             points[quads[i].p3],points[quads[i].p4], color);
}
```

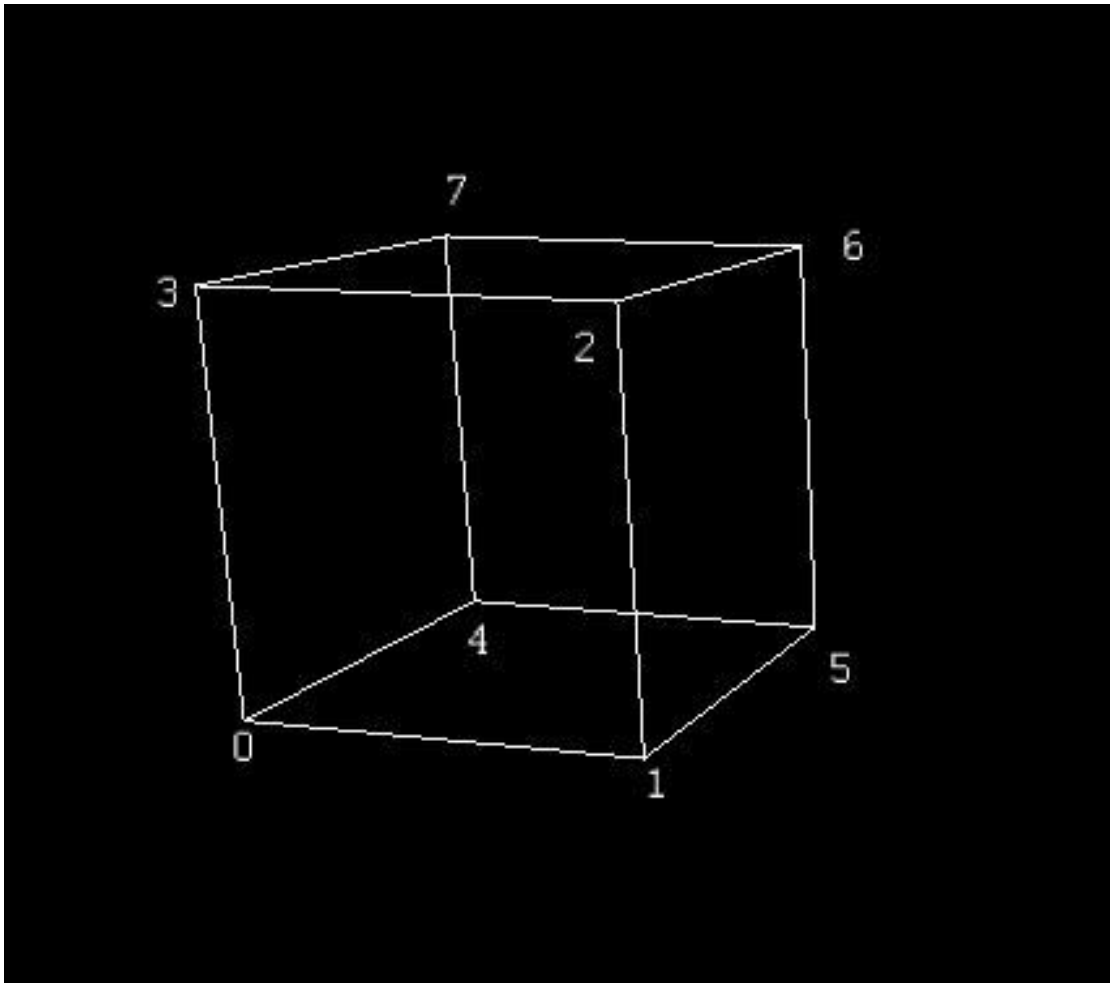
Класс Item3D

```
if ((drawtype==3)) // рисование текстурированными
треугольниками и четырехугольниками
{
 for (unsigned i = 0; i<triangles.size(); i++)
   DrawTriangleT(points[triangles[i].p1], points[triangles[i].p2],
         points[triangles[i].p3], clWhite,
      tvertex(0, 0), tvertex(0.5, 1), tvertex(1, 0), tindex);
 for (unsigned i = 0; i<quads.size(); i++)
   DrawQuadT(points[quads[i].p1], points[quads[i].p2],
         points[quads[i].p3], points[quads[i].p4],
      clWhite, tindex);
}
glPopMatrix();
}
```

Диаграмма классов



Создание куба по 8 точкам



Класс Kub

```
class Kub : public Item3D
{
public:
    Kub(vertex3 pos, float size, int drawtype, int tindex);
    Kub(vertex3 pos, float size, color3 color, int drawtype, int
    tindex);
    Kub(vertex3 pos, float size, color3 color, vertex3 angle, int
    drawtype, int   tindex);
    void Init();
};
```

Класс Kub

```
Kub::Kub(vertex3 pos, float size, int drawtype, int
    tindex):Item3D()
{
    SetPos(pos);
    SetSize(size);
    this->tindex = tindex;
    this->drawtype = drawtype;
    Init();
}
```

Класс Kub

```
Kub::Kub(vertex3 pos, float size, color3 color, int drawtype, int
    tindex) :Item3D()
{
    SetPos(pos);  SetSize(size);     SetColor(color);
    this->tindex = tindex;
    this->drawtype = drawtype;
    Init();
}
```

Класс Kub

```
Kub::Kub(vertex3 pos, float size, color3 color, vertex3 angle, int
    drawtype, int tindex) :Item3D()
{
    SetPos(pos);
    SetSize(size);
    SetColor(color);
    SetAngle(angle);
    this->tindex = tindex;
    this->drawtype = drawtype;
    Init();
}
```

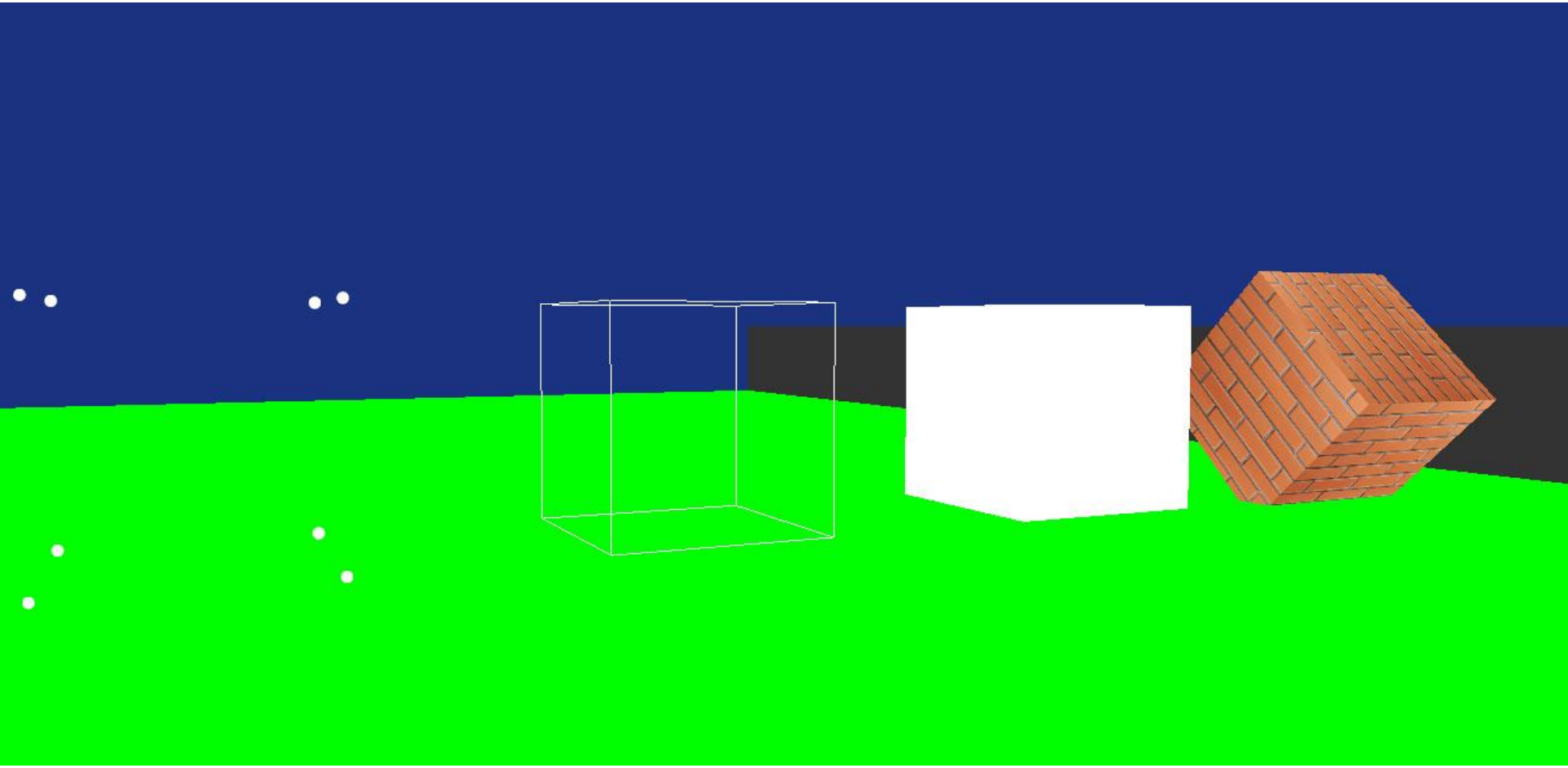
Класс Kub

```
void Kub::Init()
{
GLdouble x0 = pos.x-size/2, y0 = pos.y-size/2, z0 = pos.z-size/2;
GLdouble dx = size, dy = size, dz = size;

points.push_back(vertex(x0, y0, z0));
points.push_back(vertex(x0 + dx, y0, z0));
points.push_back(vertex(x0 + dx, y0 + dy, z0));
points.push_back(vertex(x0, y0 + dy, z0));

points.push_back(vertex(x0, y0, z0 + dz));
points.push_back(vertex(x0 + dx, y0, z0 + dz));
points.push_back(vertex(x0 + dx, y0 + dy, z0 + dz));
points.push_back(vertex(x0, y0 + dy, z0 + dz));
```

Класс Kub

```
lines.push_back(line(0, 1)); lines.push_back(line(1, 2));
lines.push_back(line(2, 3)); lines.push_back(line(0, 3));
lines.push_back(line(4, 5)); lines.push_back(line(5, 6));
lines.push_back(line(6, 7)); lines.push_back(line(4, 7));
lines.push_back(line(0, 4)); lines.push_back(line(1, 5));
lines.push_back(line(2, 6)); lines.push_back(line(3, 7));


quads.push_back(quad(0, 3, 2, 1));
quads.push_back(quad(1, 2, 6, 5));
quads.push_back(quad(5, 6, 7, 4));
quads.push_back(quad(4, 7, 3, 0));
quads.push_back(quad(2, 3, 7, 6));
quads.push_back(quad(1, 0, 4, 5));
}
```

Класс Kub



Класс Shere

```
class Shere : public Item3D
{
public:
    Shere(vertex3 pos, float size, int drawtype, int tindex);
    Shere(vertex3 pos, float size, color3 color, int drawtype, int
    tindex);
    Shere(vertex3 pos, float size, color3 color, vertex3 angle, int
    drawtype, int tindex);
    void Init();
};
```

Класс Shere

```
Shere::Shere(vertex3 pos, float size, int drawtype, int
    tindex):Item3D()
{
    SetPos(pos);
    SetSize(size);
    this->tindex = tindex;
    this->drawtype = drawtype;
    Init();
}
```

Класс Shere

```
Shere::Shere(vertex3 pos, float size, color3 color, int drawtype,
    int tindex) :Item3D()
{
    Item3D();
    SetPos(pos);
    SetSize(size);
    SetColor(color);
    this->tindex = tindex;
    this->drawtype = drawtype;
    Init();
}
```

Класс Shere

```
Shere::Shere(vertex3 pos, float size, color3 color, vertex3 angle,
    int drawtype, int tindex) :Item3D()
{
    Item3D();
    SetPos(pos);
    SetSize(size);
    SetColor(color);
    SetAngle(angle);
    this->tindex = tindex;
    this->drawtype = drawtype;
    Init();
}
```

Класс Shere

```
void Shere::Init()
{
    int i = 0, j, l, k = 0, q = 0; foat a, b;  float R = size / 2;
    int N = 20; float da = 2 * 3.1415f / (N); float x, y, z;
//  формирование точек
for (l = 0; l <= N; l++)
 for (j = 0; j < N; j++)
  {    a = j*da;      b = l*da / 2;
    x = R * sin(a)*sin(b);
    y =  - R * cos(b);
    z = R * cos(a)*sin(b);
    points.push_back(vertex(x + pos.x, y + pos.y, z + pos.z));
  }
```

Класс Shere

```
//  формирование горизонтальных линий
for (I = 0; I <N; I++)
{
     for (j = 0; j < N - 1; j++)
     {
          lines.push_back(line(j + I*N, j + 1 + I*N));
    }
    lines.push_back(line(I*N, N - 1 + I*N));
}
//  формирование вертикальных линий
for (I = 0; I < N - 1; I++)
    for (j = 0; j < N; j++)
     lines.push_back(line(j + I*N, j + N + I*N));
```

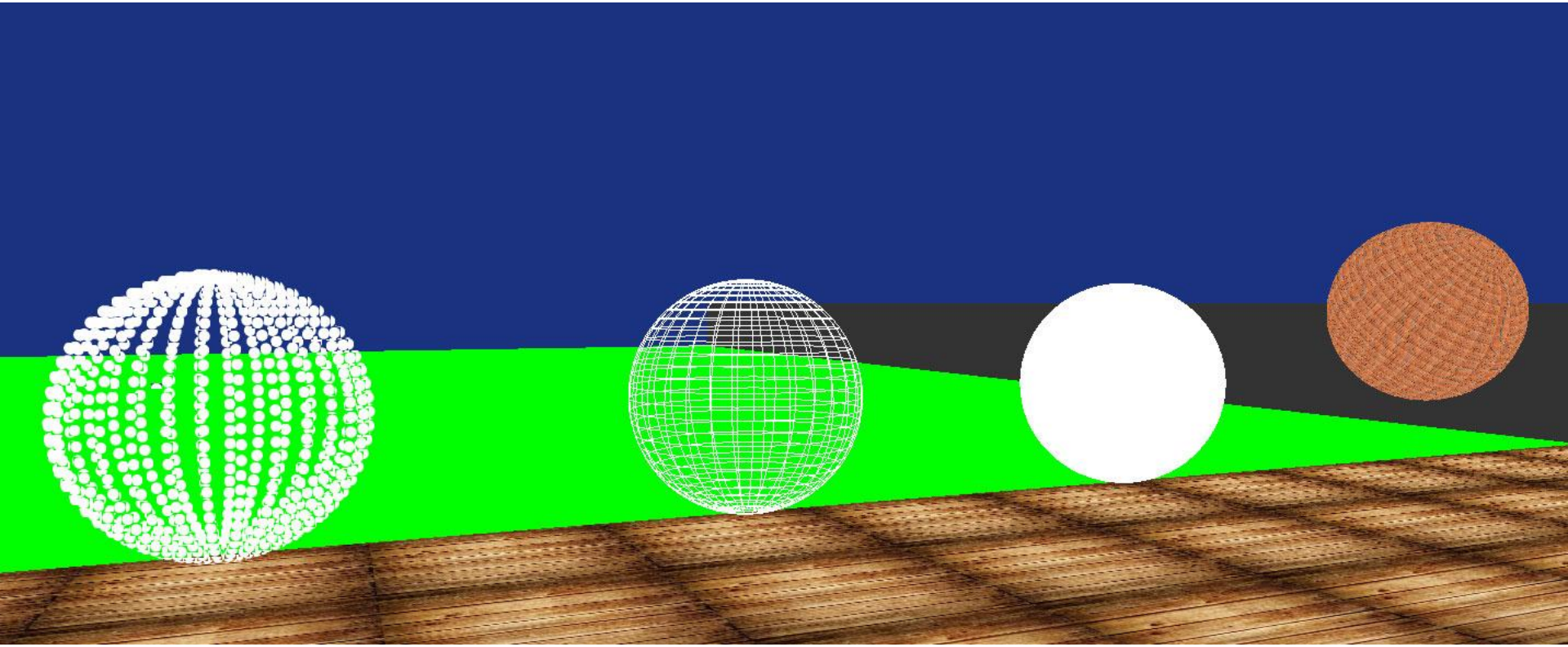
Класс Shere

```
//  формирование полигонов
float dy = 1.f / N;  float dx = 1.f / N;
for (j = 0; j < N; j++)
{
  for (l = 0; l < N; l++)
  {
    intp1 = l + j*N, p2 = l + N + j*N,
    p3 = l + N + 1 + j*N,p4 = l + 1 + j*N;
    if (l == N - 1)  {  p3 -= N;  p4 -= N;   }

    quads.push_back(quad(p1, p2, p3, p4));
```

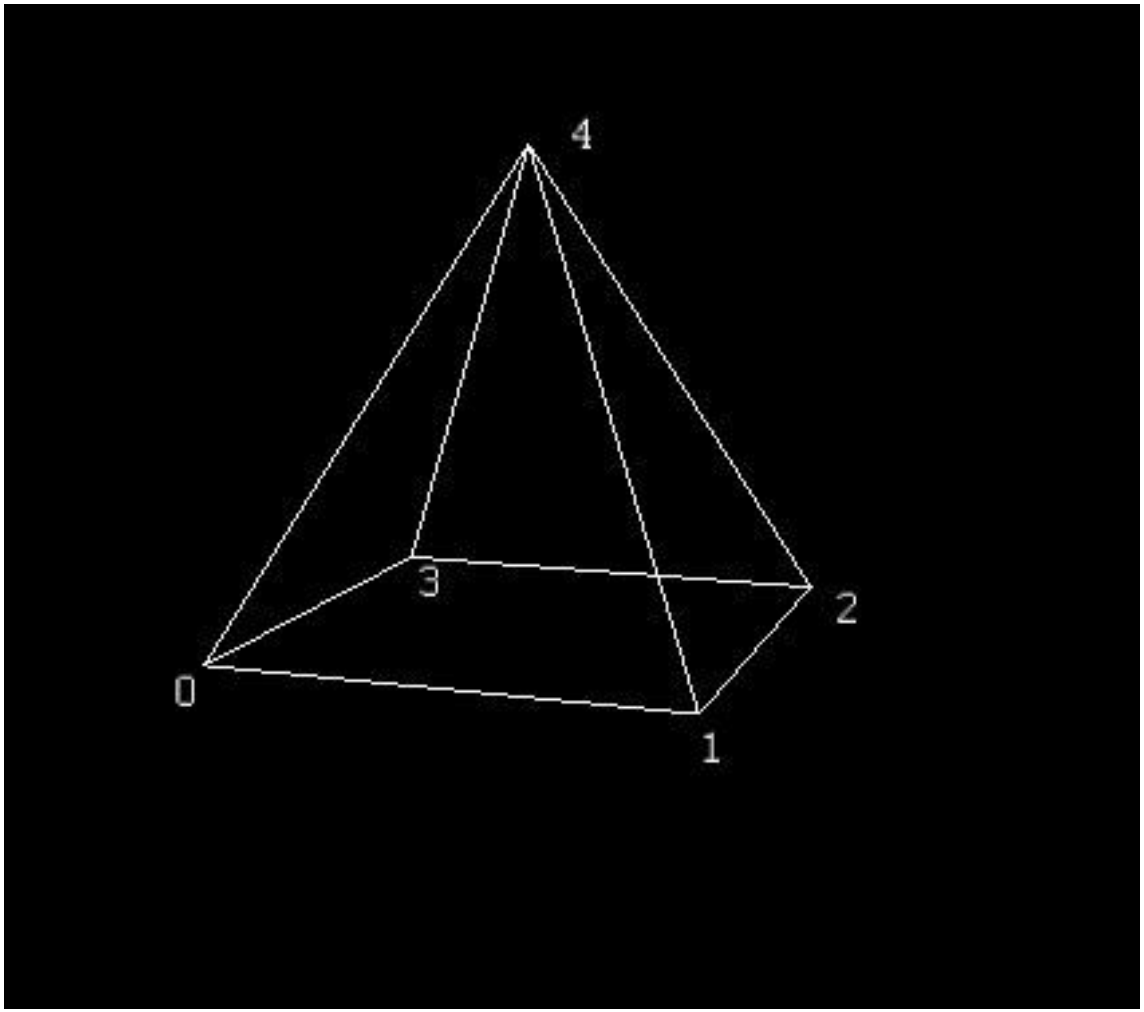
Класс Shere

```
    tvertex2 tv1 = tvertex(l*dx, j*dy);
    tvertex2 tv2 = tvertex(l*dx, j*dy + dy);
    tvertex2 tv3 = tvertex(l*dx + dx, j*dy + dy);
    tvertex2 tv4 = tvertex(l*dx + dx, j*dy);
    tvertexs.push_back(texture(tv1, tv2, tv3, tv4, tindex));
    }
}
```

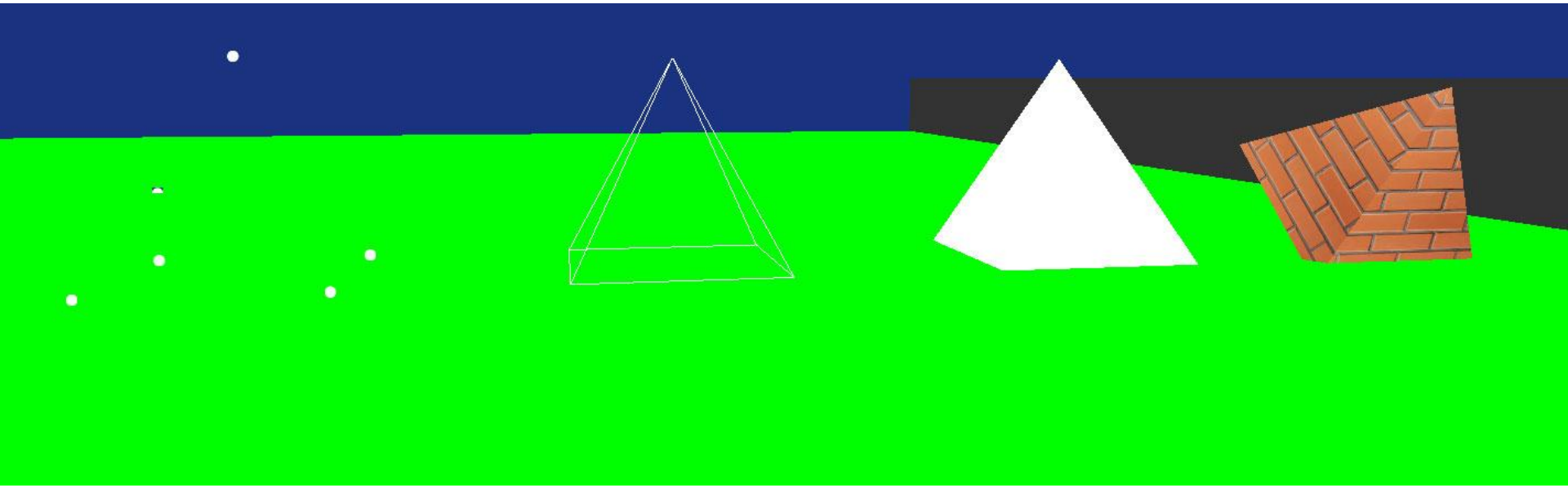
Класс Sphera



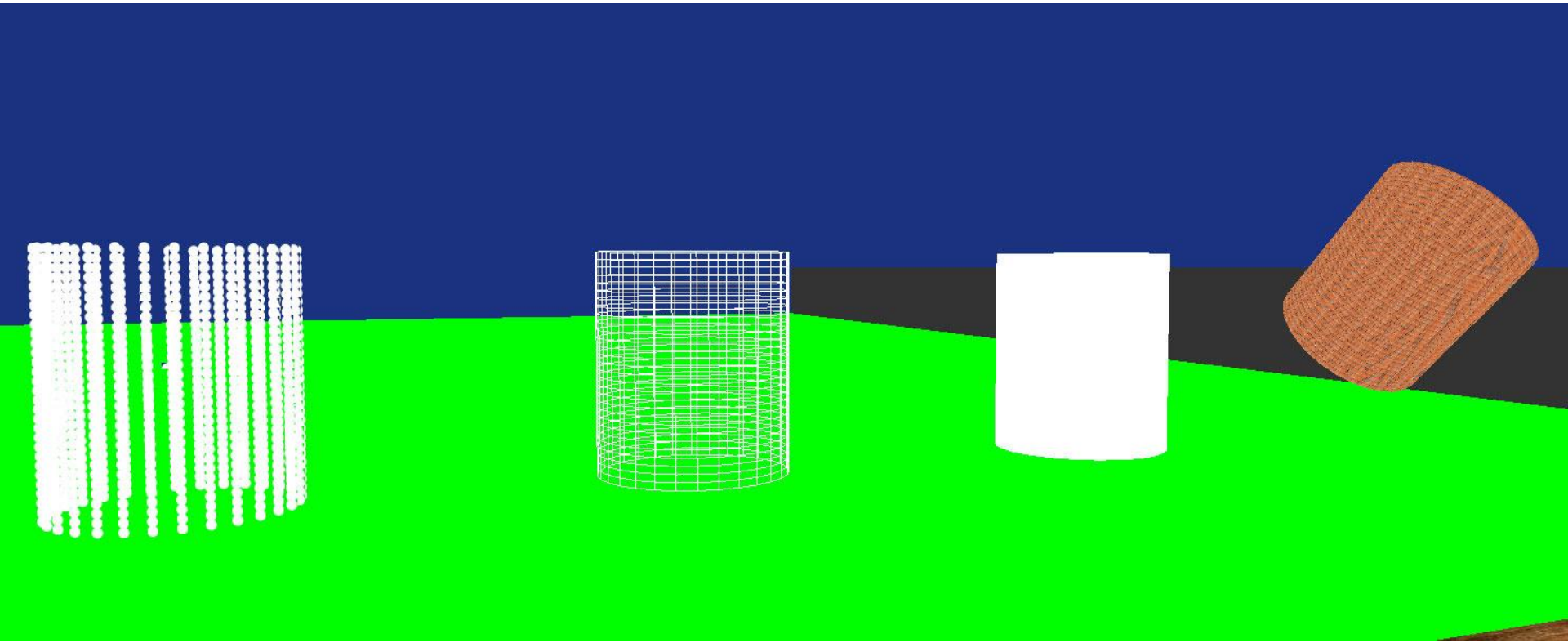
Создание пирамиды по 5 точкам



Класс Piramida



Класс Cilindr



Класс TexturesList

```
class TexturesList
{
    vector<string> textures_names;
    vector<GLuint> textures_index;
public:
    void LoadTexture(string name, string type = "bmp");
    GLuint getTextureIndex(string name);
};
```

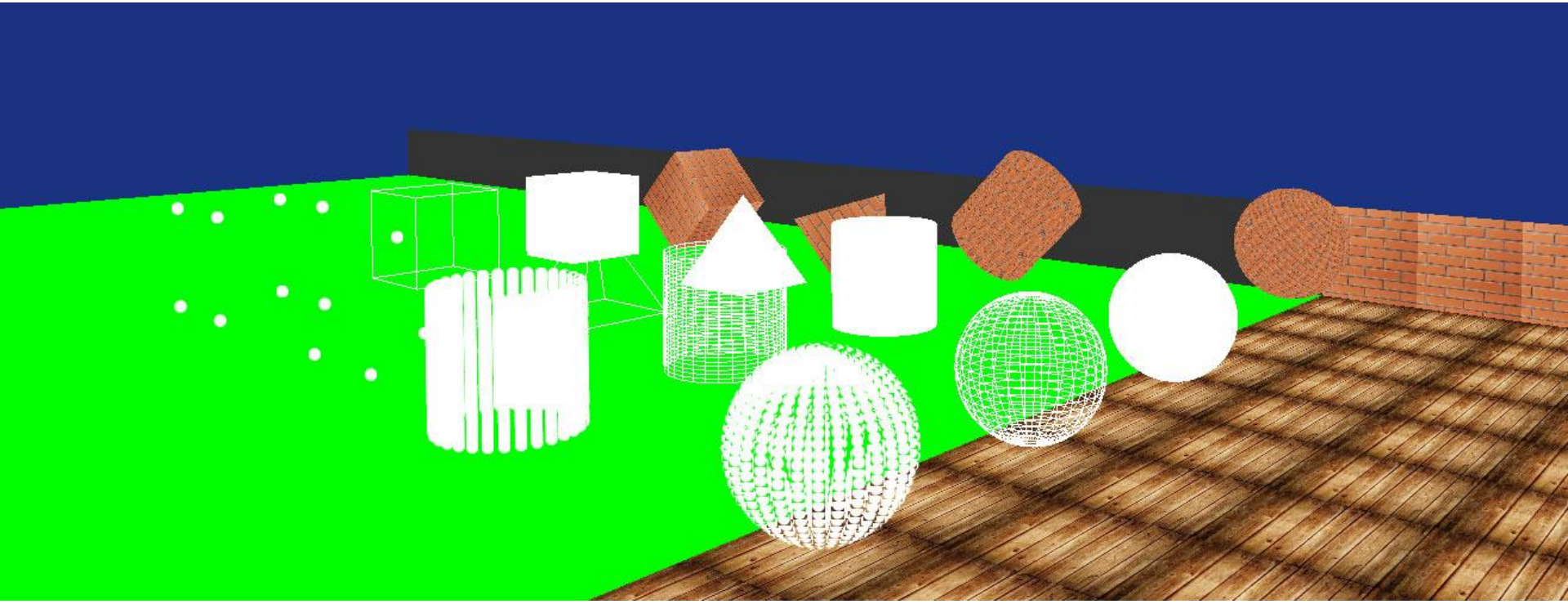
Класс TexturesList

```
void TexturesList::LoadTexture(string name, string type)
{
    int i = textures_index.size();
    GLuint value;
    if (strcmp(type.c_str(), "bmp") ==0)
     LoadBMP(&value, (char *)name.c_str());
    if (strcmp(type.c_str(), "tga") == 0)
          LoadTGA(&value, (char *)name.c_str());
    textures_index.push_back(value);
    textures_names.push_back(name);
}
```

Класс TexturesList

```
GLuint TexturesList::getTextureIndex(string name)
{
    if (textures_names.size()>0)
        for (unsigned i = 0; i<textures_names.size(); i++)
            if (textures_names[i].compare(name)==0)
                return textures_index[i];

    return -1;
}
```

Класс SCENE



Класс SCENE

```
static TexturesList * textures;
using namespace std;
class SCENE
{
    std::vector<Item *> items;
public:
    SCENE();
    ~SCENE();
    void Init();
    void Draw();
    static GLuint getTextureIndex(string name);
    void InitLight();
    void SetLight();
};
```

Класс SCENE

```
SCENE::SCENE()
{
    Init();
}

SCENE::~SCENE()
{
    for (unsigned i = 0; i<items.size(); i++)
     delete items[i];
    items.clear();
}
```

Класс SCENE

```
void SCENE::Init()
{
    textures = new TexturesList();

    textures->LoadTexture("wall1.bmp");
    textures->LoadTexture("wall2.bmp");
    textures->LoadTexture("Grass32.bmp");

    items.push_back(new Line(vertex(-10, 0, 0), vertex(10, 0, 0),
     color(1, 1, 1), 1));
    items.push_back(new Line(vertex(0, -10, 0), vertex(0, 10, 0),
     color(1, 1, 1), 1));
    items.push_back(new Line(vertex(0, 0, -10), vertex(0, 0, 10),
     color(1, 1, 1), 1));
```

Класс SCENE

```
// points
items.push_back(new Point(vertex(0, 0, 0), color(1, 1, 1), 10));
items.push_back(new Point(vertex(0, 1, 0), color(1, 0, 0), 5));
items.push_back(new Point(vertex(0, -1, 0), color(1, 0, 0), 5));
items.push_back(new Point(vertex(1, 0, 0), color(0, 1, 0), 5));
items.push_back(new Point(vertex(-1, 0, 0), color(0, 0, 1), 5));

// triangles
items.push_back(new Triangle(vertex(-0.5, 0.2, 0), vertex(0,
    0.5, 0), vertex(0.5, 0.2, 0), color(0, 1, 0)));
items.push_back(new Triangle(vertex(-0.5, -0.2, 0), vertex(0, -
        0.5, 0), vertex(0.5, -0.2, 0), color(1, 0, 0), color(0, 1, 0),
        color(0, 0, 1)));
```

Класс SCENE

```
// quads
Quad * quad = new Quad(vertex(-2, -2, 0), vertex(-2, -1, 0),
    vertex(-1, -1, 0), vertex(-1, -2, 0), color(1, 0, 0));
quad->SetAngle(vertex(0, 0, 45));
items.push_back(quad);
items.push_back(new Quad(vertex(-2, 2, 0), vertex(-2, 1, 0),
    vertex(-1, 1, 0), vertex(-1, 2, 0), color(1, 0, 0), color(0, 1, 0),
    color(0, 0, 1), color(0, 1, 1)));
```

Класс SCENE

```
// quadst
items.push_back(new QuadT(vertex(1, 1, 0), vertex(1, 2, 0),
    vertex(2, 2, 0), vertex(2, 1, 0), color(1, 1, 1), "wall1.bmp",
    tvertex(0, 0), tvertex(0, 1), tvertex(1, 1), tvertex(1, 0)));
items.push_back(new QuadT(vertex(1, 2, 0), vertex(1, 3, 0),
    vertex(2, 3, 0), vertex(2, 2, 0), color(1, 0, 0), "wall1.bmp",
    tvertex(0, 0), tvertex(0, 0.5), tvertex(0.5, 0.5), tvertex(0.5, 0)));


items.push_back(new Kub(vertex(1, 0, 0), 1, color(1, 1, 1), 1, 0));
items.push_back(new Shere(vertex(3, 0, 0), 2, color(1, 1, 1), 1,
    0));
}
```

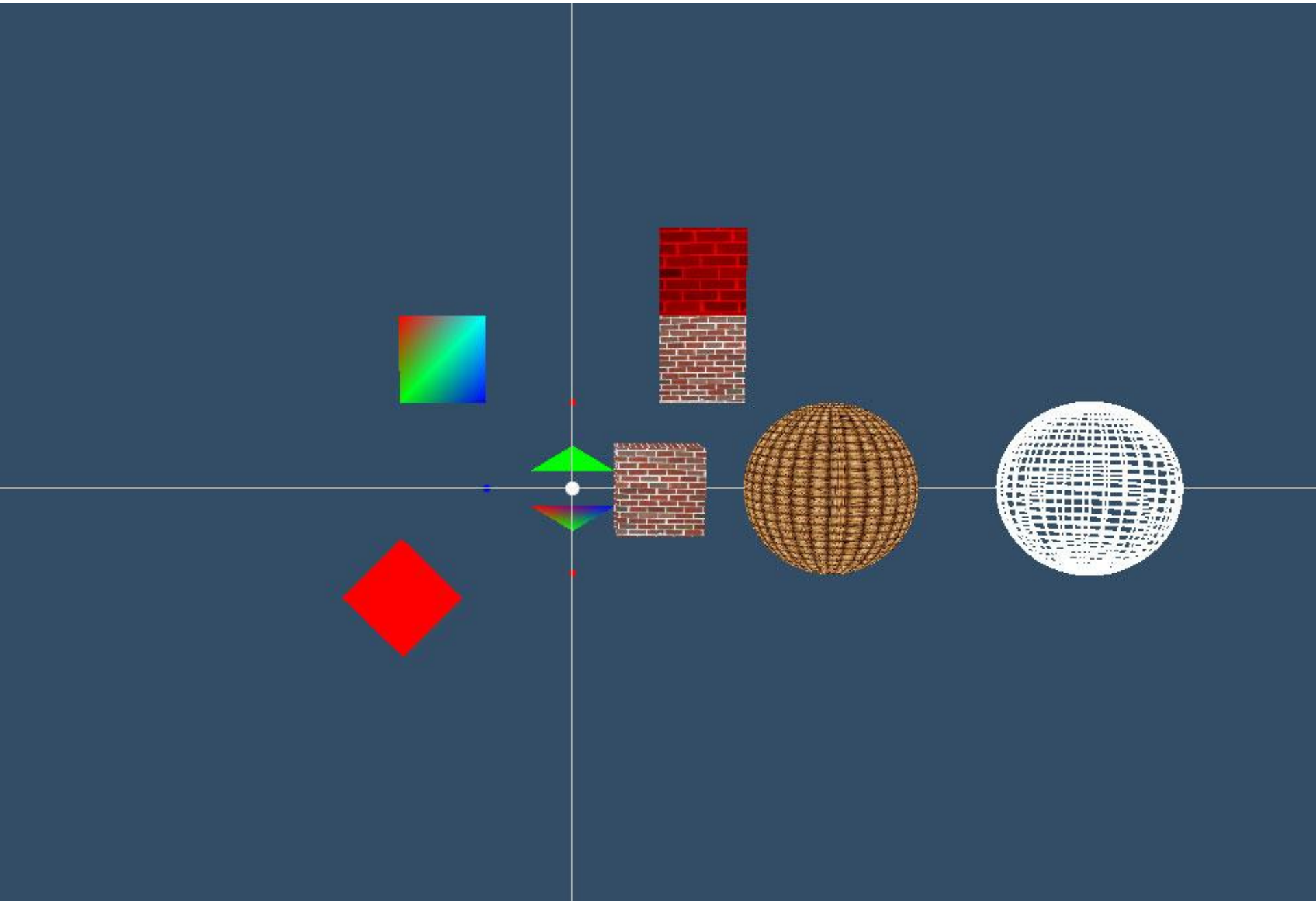
Класс SCENE

```
GLuint SCENE::getTextureIndex(string name)
{
    return textures->getTextureIndex(name);
}
```

Класс SCENE

```
void SCENE::Draw()
{
    for (unsigned i = 0; i<items.size(); i++)
        items[i]->Draw();

    glLoadIdentity();
    glTranslated(0, 3, 0);

    CalcFPS();
    DrawFPS();
}
```

Класс SCENE



Дополнительные структуры

```
struct texture4 { tvertex2 t1, t2, t3, t4; GLuint index; };
// Структура для текстуры с 4 текстурными
   координатами и индексом текстуры


// создание структуры текстуры
texture4 texture(tvertex2 t1, tvertex2 t2, tvertex2 t3,
   tvertex2 t4, GLuint index)
{
   texture4 tv = { t1, t2, t3, t4, index};
   return tv;
}
```

Функция рисования текстурированного четырехугольника

```
void DrawQuadT(vertex3 v1, vertex3 v2, vertex3 v3, vertex3 v4,
texture4 t,color3 color)
{   glEnable(GL_TEXTURE_2D);
    glColor3d(color.r, color.g, color.b);
    glBindTexture(GL_TEXTURE_2D, t.index);
    glBegin(GL_QUADS);
     glTexCoord2f(t.t1.x, t.t1.y);    glVertex3d(v1.x, v1.y, v1.z);
     glTexCoord2f(t.t2.x, t.t2.y);    glVertex3d(v2.x, v2.y, v2.z);
     glTexCoord2f(t.t3.x, t.t3.y);    glVertex3d(v3.x, v3.y, v3.z);
     glTexCoord2f(t.t4.x, t.t4.y);    glVertex3d(v4.x, v4.y, v4.z);
    glEnd();
glDisable(GL_TEXTURE_2D);  }
```

Класс Box

```
// текстурированная коробка с разными текстурами на 6ти
    гранях
class Box : public Kub
{
    texture4 masT[6];
    vertex3 masP[8];
public:
    Box(vertex3 pos, float width, float height, float length,
    texture4 t1, texture4 t2, texture4 t3, texture4 t4, texture4 t5,
    texture4 t6);
    void Draw();
};
```

Класс Box

```
Box::Box(vertex3 pos, float width, float height, float length,
texture4 t1, texture4 t2, texture4 t3, texture4 t4, texture4 t5,
    texture4 t6):Kub(pos, width, 0, 0)
{
    masT[0] = t1;masT[1] = t2;
    masT[2] = t3;masT[3] = t4;
    masT[4] = t5;masT[5] = t6;

    masP[0] = vertex(pos.x - width / 2, pos.y - height / 2, pos.z -
              length / 2);
    masP[1] = vertex(pos.x + width/2, pos.y - height / 2, pos.z -
              length / 2);
    masP[2] = vertex(pos.x + width / 2, pos.y + height / 2, pos.z -
          length / 2);
```

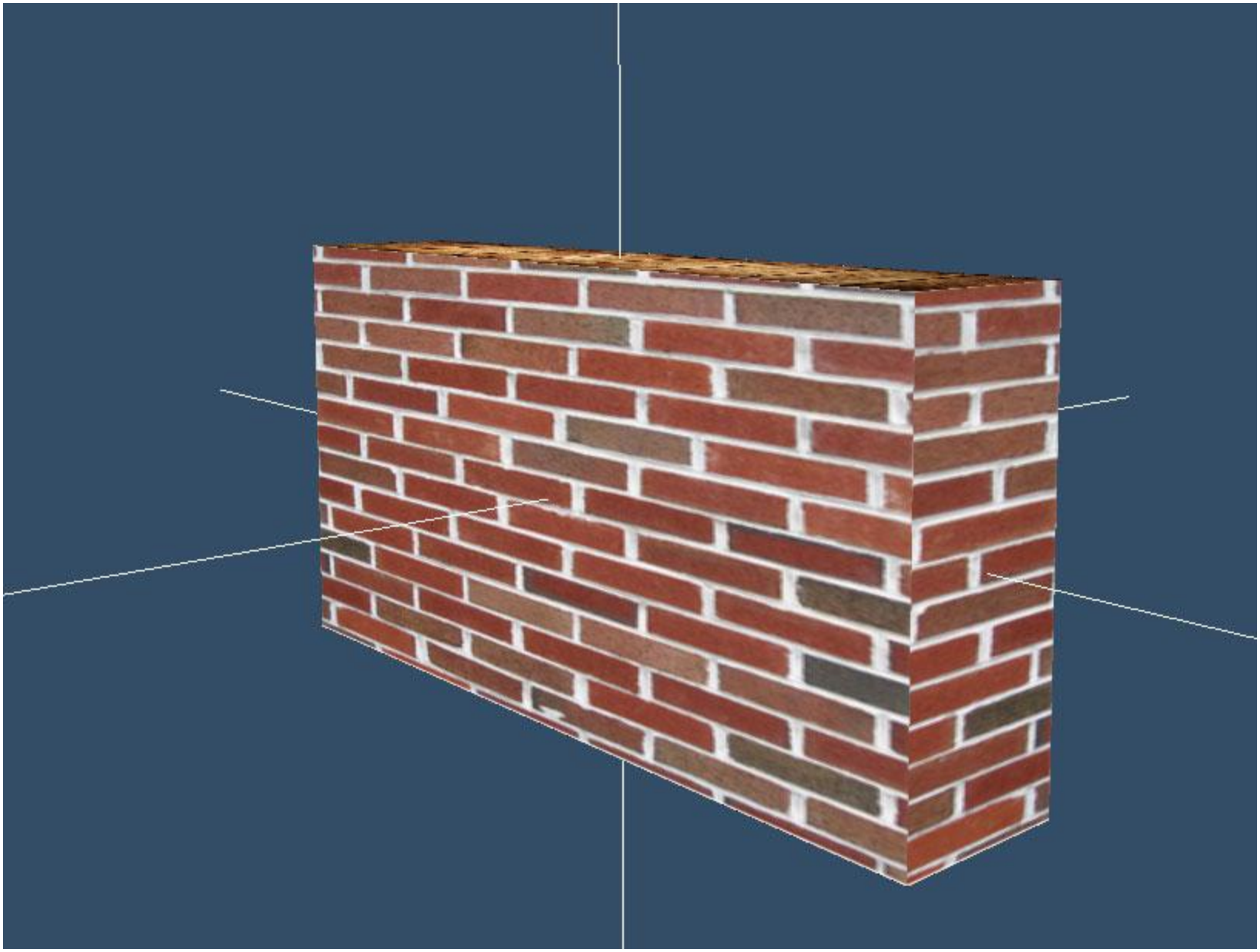
Класс Box

```
    masP[3] = vertex(pos.x - width / 2, pos.y + height / 2, pos.z -
            length / 2);
    masP[4] = vertex(pos.x - width / 2, pos.y - height / 2, pos.z +
            length / 2);
    masP[5] = vertex(pos.x + width / 2, pos.y - height / 2, pos.z +
        length / 2);
    masP[6] = vertex(pos.x + width / 2, pos.y + height / 2, pos.z +
            length / 2);
    masP[7] = vertex(pos.x - width / 2, pos.y + height / 2, pos.z +
        length / 2);
}
```

Класс Box

```
void Box::Draw()
{ // дальняя стена
    DrawQuadT(masP[0], masP[3], masP[2], masP[1], masT[0]);
    // верх
    DrawQuadT(masP[7], masP[3], masP[2], masP[6], masT[1]);
     // низ
    DrawQuadT(masP[4], masP[0], masP[1], masP[5], masT[2]);
    // передняя стена
    DrawQuadT(masP[4], masP[7], masP[6], masP[5], masT[3]);
    // левая стена
    DrawQuadT(masP[0], masP[3], masP[7], masP[4], masT[4]);
     // правая стена
    DrawQuadT(masP[1], masP[2], masP[6], masP[5], masT[5]);
}
```

Класс Box



Вычисление нормали

```
//   Возвращает вектор, перпендикулярный 2м переданным.
vertex3 Cross(vertex3 v1, vertex3 v2)
{
    vertex3 vNormal;
    vNormal.x = ((v1.y * v2.z) - (v1.z * v2.y));
    vNormal.y = ((v1.z * v2.x) - (v1.x * v2.z));
    vNormal.z = ((v1.x * v2.y) - (v1.y * v2.x));
    return vNormal;
}
//   возвращает величину нормали
float Magnitude(vertex3 vNormal)
{    return (float)sqrt((vNormal.x * vNormal.x) +
     (vNormal.y * vNormal.y) + (vNormal.z * vNormal.z));
}
```

Вычисление нормали

```
//   возвращает нормализованный вектор (с длинной 1)
vertex3 Normalize(vertex3 vNormal)
{
    float magnitude = Magnitude(vNormal);
    vNormal.x /= magnitude;
    vNormal.y /= magnitude;
    vNormal.z /= magnitude;
    return vNormal;
}
//   Возвращает вектор между 2мя точками.
vertex3 Sub(vertex3 v1, vertex3 v2)
{
    return vertex(v1.x - v2.x, v1.y - v2.y, v1.z - v2.z);
}
```

Вычисление нормали

```
//   Возвращает сумму векторов.
vertex3 Sum(vertex3 v1, vertex3 v2)
{
    return vertex(v1.x + v2.x, v1.y + v2.y, v1.z + v2.z);
}
//   Возвращает нормаль по трём точкам
vertex3 Normal(vertex3 v1, vertex3 v2, vertex3 v3)
{
    vertex3 V1 = Sub(v3, v1);    vertex3 V2 = Sub(v2, v1);
    vertex3 vNormal = Cross(V1, V2);
    vNormal = Normalize(vNormal);
    return vNormal;
}
```

Рисование четырехугольника с нормалями

```
void DrawQuadT(vertex3 v1, vertex3 v2, vertex3 v3, vertex3 v4,
tvertex2 tv1, tvertex2 tv2, tvertex2 tv3, tvertex2 tv4,
color3 color, GLuint texture)
{   glEnable(GL_TEXTURE_2D);
    glColor3d(color.r, color.g, color.b);
    glBindTexture(GL_TEXTURE_2D, texture);
    glBegin(GL_QUADS);
     vertex3 n = Normal(v1, v2, v3);
     glNormal3d(n.x, n.y, n.z);
     glTexCoord2f(tv1.x, tv1.y);   glVertex3d(v1.x, v1.y, v1.z);
     glTexCoord2f(tv2.x, tv2.y);   glVertex3d(v2.x, v2.y, v2.z);
     glTexCoord2f(tv3.x, tv3.y);   glVertex3d(v3.x, v3.y, v3.z);
     glTexCoord2f(tv4.x, tv4.y);   glVertex3d(v4.x, v4.y, v4.z);
    glEnd();
    glDisable(GL_TEXTURE_2D);
}
```