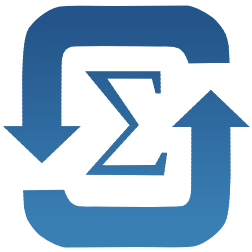
Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Прикладной математики

Лабораторная работа №2
по дисциплине «Комьютерная графика»

Трехмерная визуализация в режиме реального времени



Факультет:	ПМИ
Группа:	ПМ-63
Студент:	Шепрут И.И.
Вариант:	-
Преподаватель:	Задорожный А.Г.

Новосибирск
2019

1 Цель работы

Ознакомиться с методом тиражирования сечений (основным способом задания полигональных моделей) и средствами трехмерной визуализации (системы координат, источники света, свойства материалов).

2 Постановка задачи

1. Считывать из файла (в зависимости от варианта):
1.1. 2D-координаты вершин сечения (считающегося выпуклым);
1.2. 3D-координаты траектории тиражирования;
1.3. параметры изменения сечения.
2. Построить фигуру в 3D по прочитанным данным.
3. Включить режимы:
3.1. буфера глубины;
3.2. двойной буферизации;
3.3. освещения и материалов.
4. Предоставить возможность показа:
4.1. каркаса объекта;
4.2. нормалей (например, отрезками);
4.3. текстур, «обернутых» вокруг фигуры.
5. Предоставить возможность переключения между режимами ортографической и перспективной проекции.
6. Обеспечить навигацию по сцене с помощью модельно-видовых преобразований, сохраняя положение источника света.
7. Предоставить возможность включения/выключения режима сглаживания нормалей.

Вариант: **Рендеринг порталов.**

3 Реализованные функции

- **Рендеринг порталов.**
  - Порталы рисуются во всех возможных граничных случаях. Никаких перекрытий или артефактов во время рендеринга не появится.
- **Считывание произвольной сцены из многоугольников, с анимаций, из `json` файла.**
  - Поддержка задания анимаций в сцене.
- **Поддержка текстур.**
- **Рисование невыпуклых многоугольников.**
- Вращение камеры вокруг сцены при помощи мыши.
  - Так же имеется приближение/отдаление с помощью колесика мыши.
- Имеется меню на все основные функции.
- Рендеринг с учетом точечных источников света.

3.1 Для достижения всего этого, были использованы:

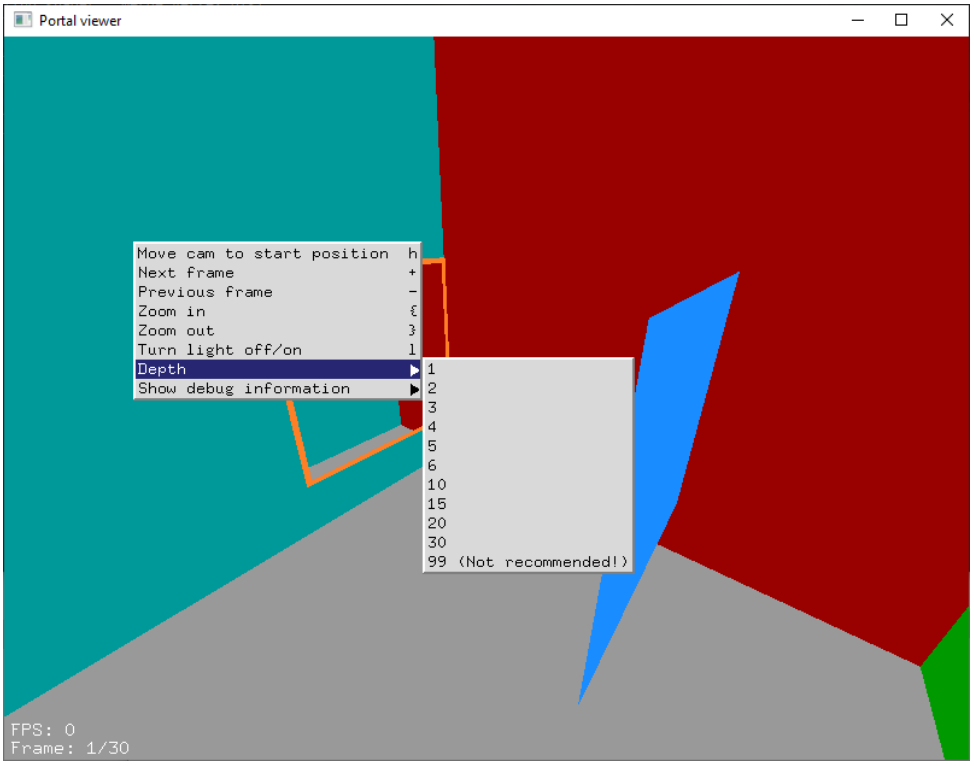
Библиотека `glm` для работы с матрицами и векторами.
- `ClipPlane` для удаления невидимых частей при рендеринге сцены «внутри» портала.

- Собственный расчет `ClipPlane` для определения того, необходимо рисовать портал или нет, так как рендеринг портала - самая дорогая по времени операция.
- Собственный расчет определения лицевых граней для рендеринга фронтальной и задней стороны портала. Опять же в целях оптимизации.
- Расчет пересечений спроецированных порталов с помощью библиотеки `clipper` при рендеринге порталов более глубокого уровня, для того чтобы определить порталы, которые точно не будут видны на экране.
- `Framebuffer`'ы с цветом и глубиной, для того, чтобы рисовать сцену каждого портала в отдельном буфере.
- Шейдеры, для того, чтобы при помощи них объединять `Framebuffer`'ы, или рисовать их на экране.
- Библиотека `stb_image.h` для считывания изображений для текстур.
- Библиотека `nlohmann/json` для считывания и записи сцен в `json`.
- Методы `gluTess`, для получения множества выпуклых примитивов из невыпуклого многоугольника.

4 Пример реализованных функций

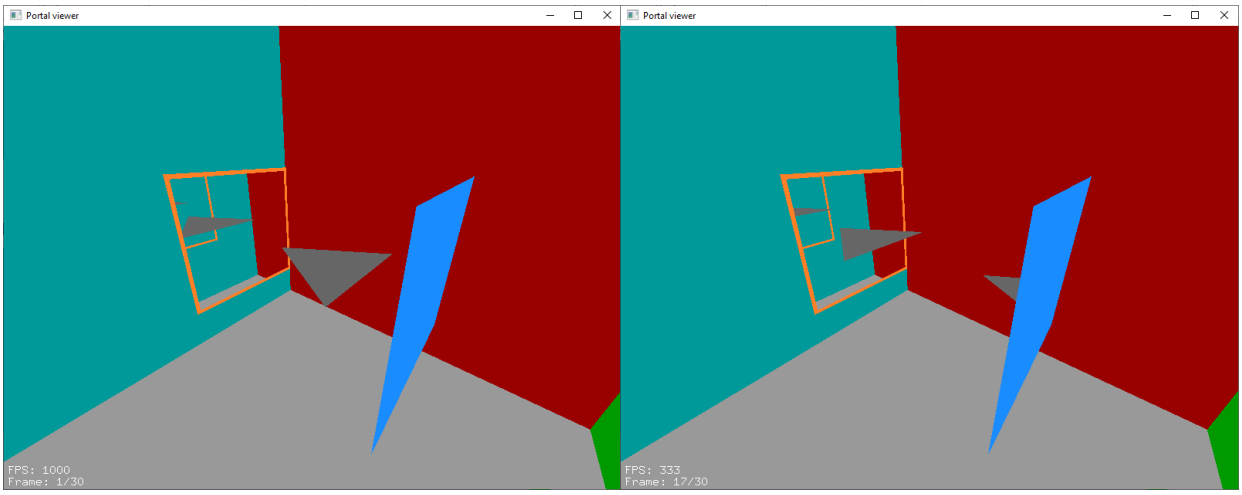
4.1 Меню

В меню так же с отступом написано как воспользоваться данной возможностью при помощи клавиатуры, а конкретно, там сказана клавиша, реализующая это действие.

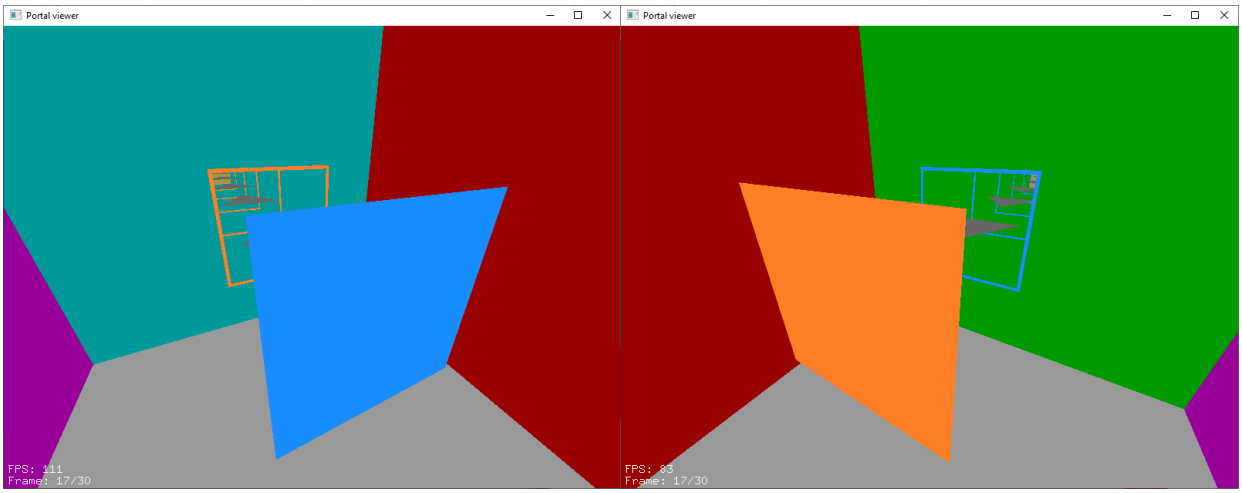


4.2 Анимация

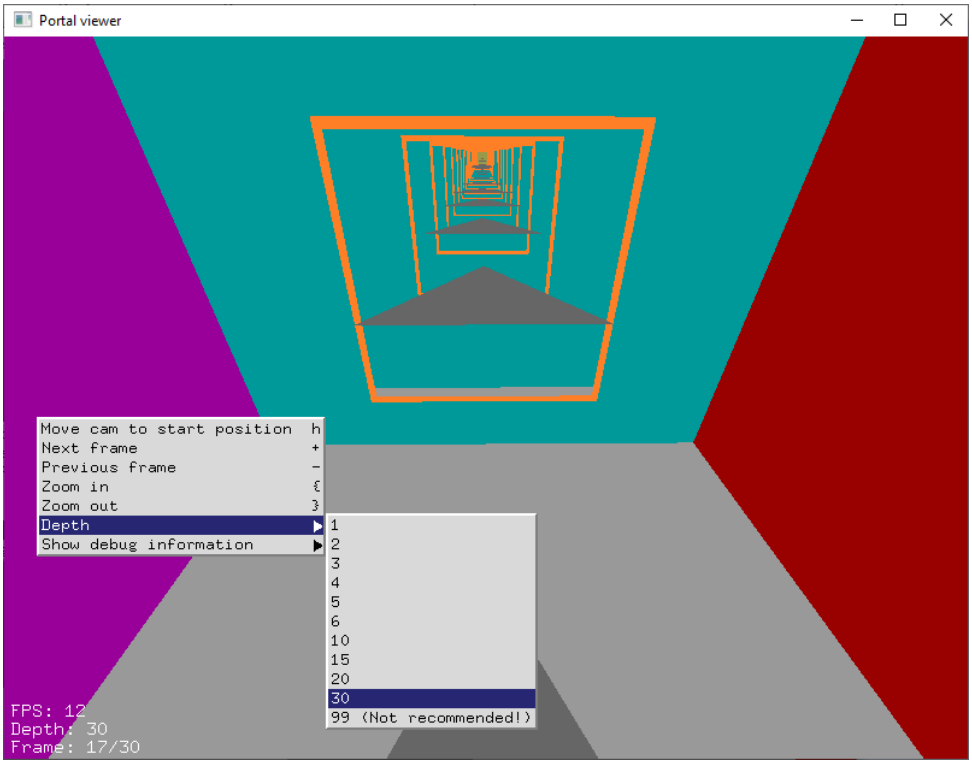
Внизу можно увидеть номер текущего кадра во всей анимации.



4.3 Простая сцена с порталами

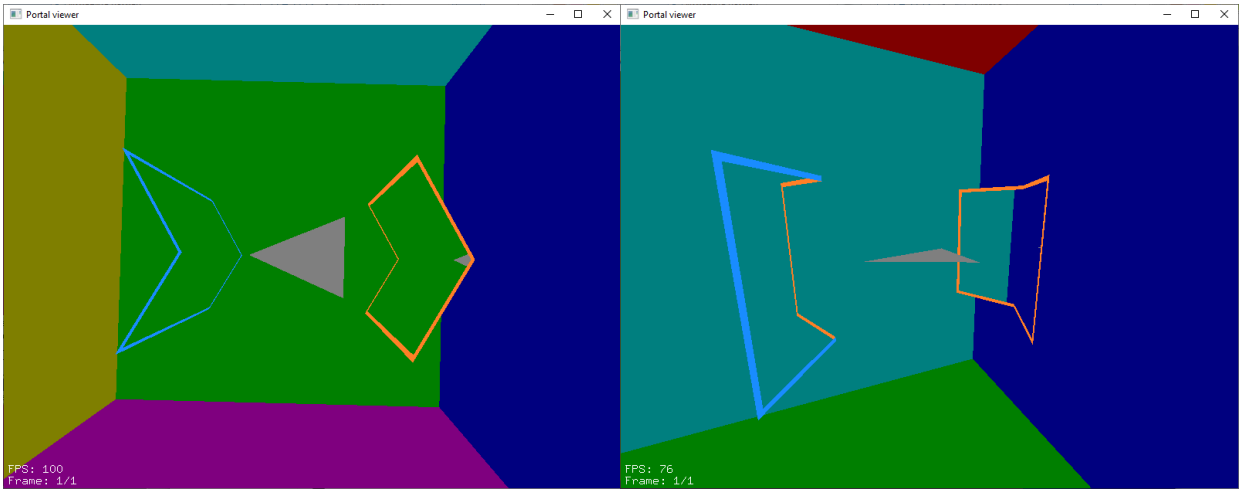


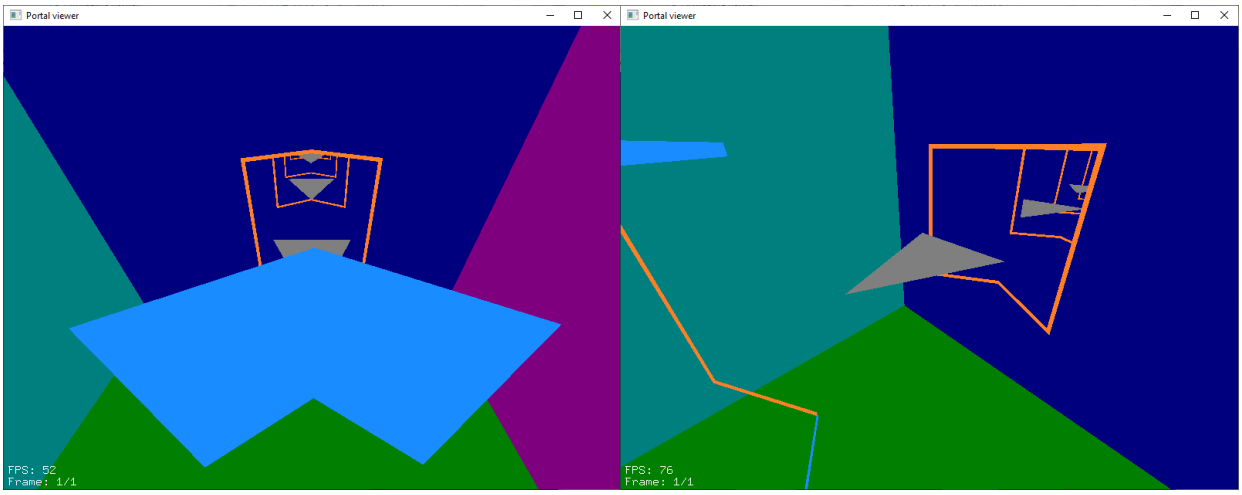
4.4 Задание большой глубины



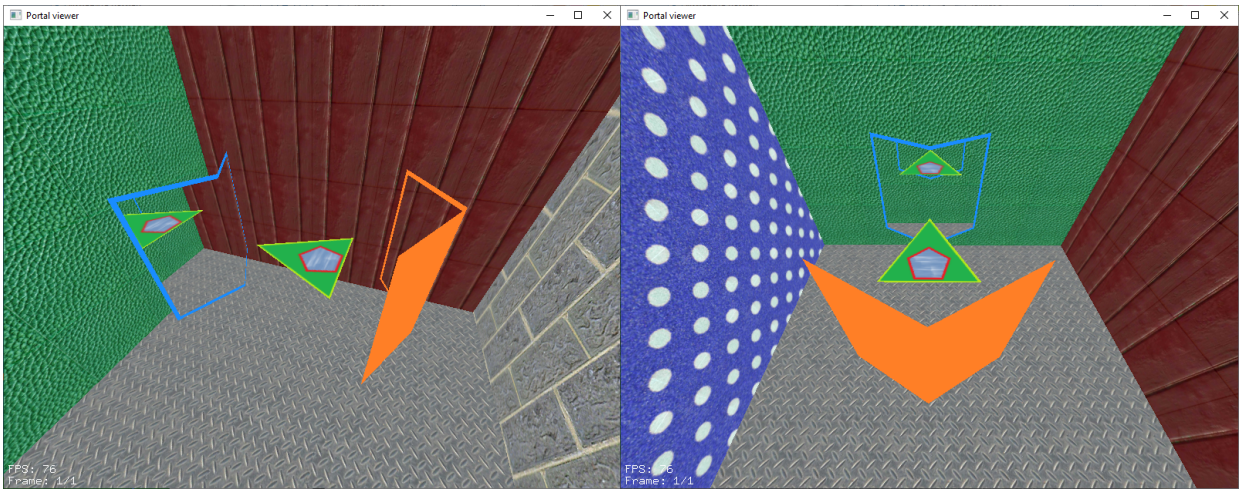
4.5 Объёмный портал

Примечание: объёмный портал — это портал, полученный соединением нескольких обычных порталов таким образом, что сохраняется непрерывность и целостность пространства. Самое главное в этом тесте, что через портал видно другую часть портала, а через неё исходную сцену.

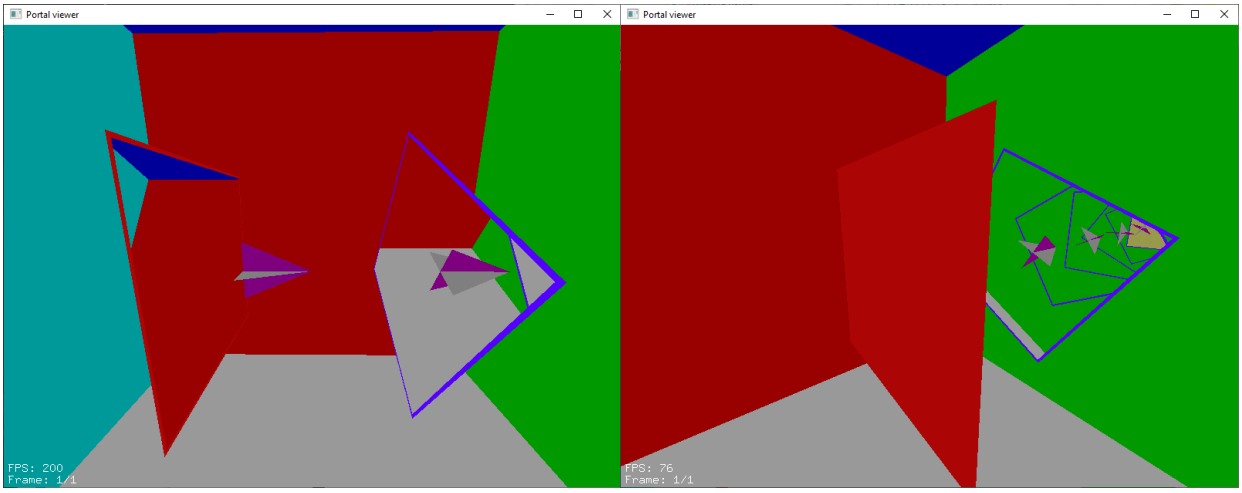




4.6 Использование текстур

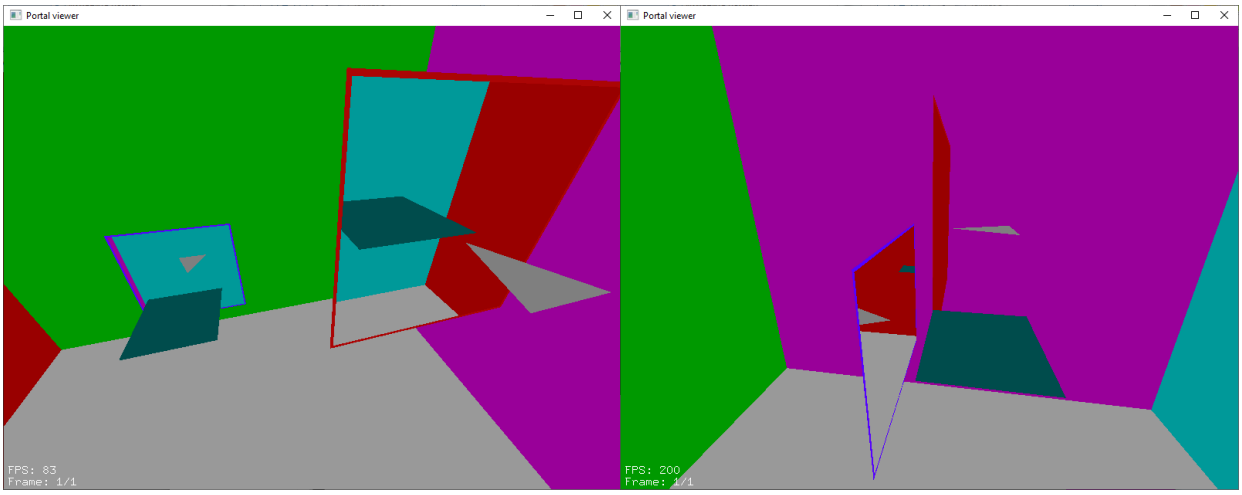


4.7 Наклоненный портал



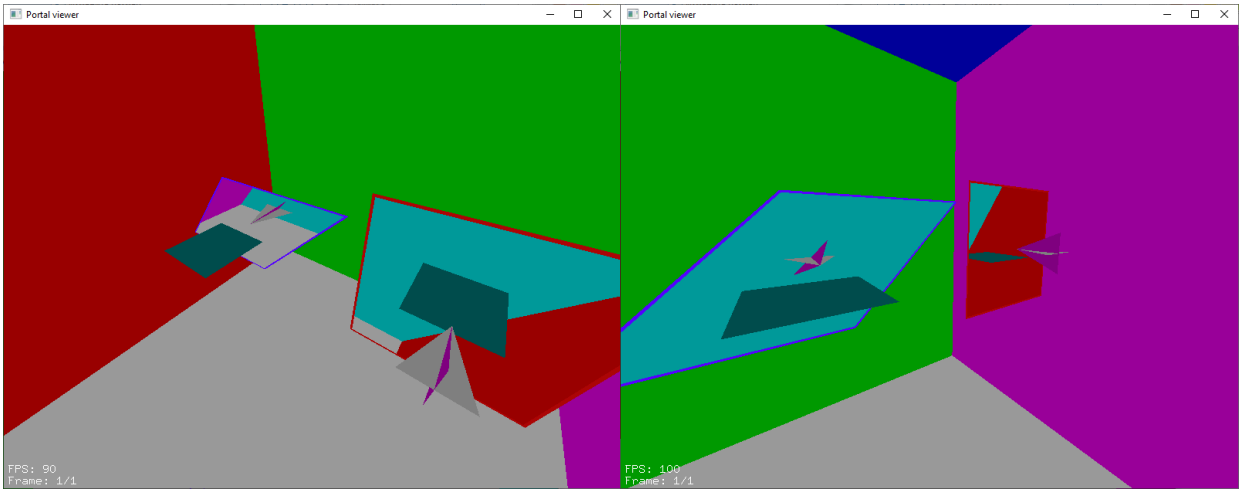
4.8 Портал, изменяющий масштаб мира

Одна часть данного портала уменьшена по сравнению с другой, получается, всё, что войдет в этот портал, будет уменьшено, а что войдет в другой, увеличено. На данной сцене можно видеть, что в одном портале полигон уменьшается, а на другом увеличивается.



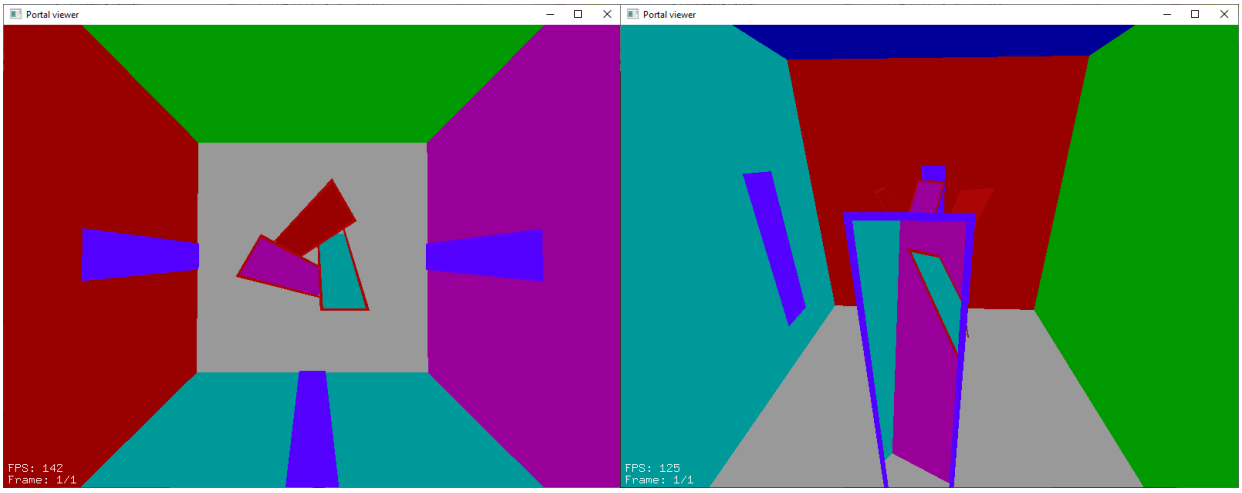
4.9 Портал, изменяющий наклон мира

Аналогично предыдущему, только здесь одна часть наклонена относительно другой, и получается, что при просмотре из одного портала в другой, наклоняется весь мир.

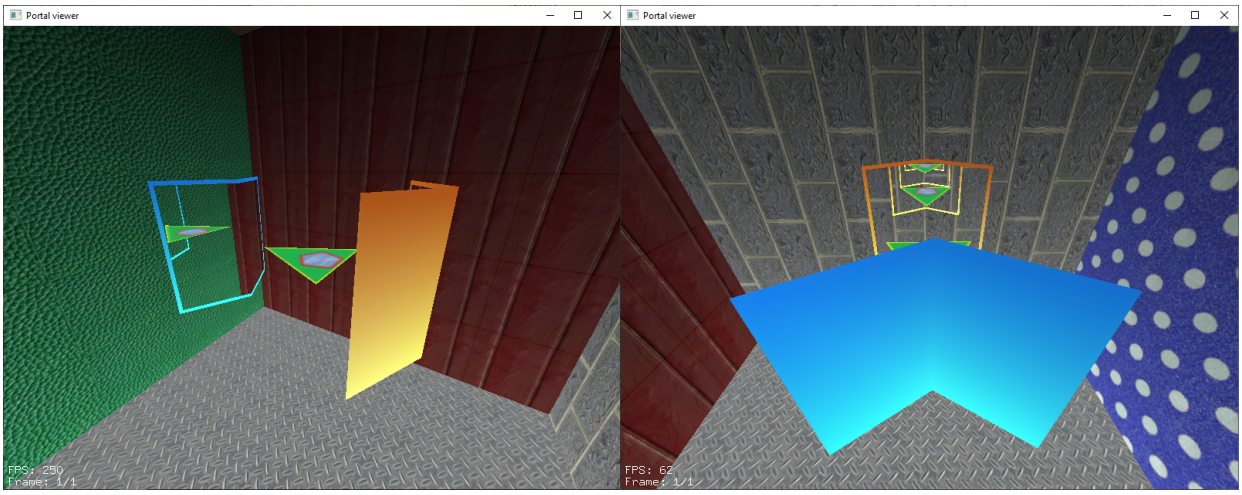


4.10 Пропеллер из порталов

Этот тест призван показать, что никаких пересечений сцены или рендеринга между порталами нет, и они корректно работают с буфером глубины. При рендеринге порталов, считается, что вся сцена, находящаяся внутри него, рисуется как текстура.



4.11 Работа освещения



5 Код программы

5.1 Файлы заголовков

FILE **fragment.h**

```
#pragma once

#include <GL/glew.h>
#include <glm/glm.hpp>
#include <vector>

//--------------------------------------------------------------------------
struct Fragment
{
    int begin;
    std::vector<glm::vec3> vertices;
};

struct TexFragment
{
    int begin;
    std::vector<glm::vec3> vertices;
    std::vector<glm::vec2> tex_coords;
};

void drawFragment(const Fragment& f);
void drawFragments(const std::vector<Fragment>& f);

void drawFragment(const TexFragment& f);
void drawFragments(const std::vector<TexFragment>& f);

class Fragmentator
{
public:
    static std::vector<Fragment> fragmentize(const std::vector<glm::vec4>&
    ↪  polygon);

    static std::vector<TexFragment> fragmentize(const std::vector<glm::vec4>&
    ↪  polygon, const std::vector<glm::vec4>& tex_coords);
```

```
private:
    static std::vector<Fragment> fragments;
    static std::vector<TexFragment> texFragments;

    static void __stdcall tessBegin1(GLenum which);
    static void __stdcall tessEnd1();
    static void __stdcall tessVertex1(const GLvoid *data);
    static void __stdcall tessError1(GLenum errorCode);

    static void __stdcall tessBegin2(GLenum which);
    static void __stdcall tessEnd2();
    static void __stdcall tessVertex2(const GLvoid *data);
    static void __stdcall tessError2(GLenum errorCode);
};
```

FILE **framebuffer.h**

```
#pragma once

#include <GL/glew.h>
#include <glm/glm.hpp>
#include <vector>
#include <prtl_vis/fragment.h>

class FrameBufferDrawer;

//--------------------------------------------------------------------------
class FrameBuffer
{
public:
    FrameBuffer(int width, int height);
    ~FrameBuffer();

    void activate(bool isClear = true) const;
    void disable(bool isClear = true) const;

    GLuint getFrameBuffer(void) const;
    GLuint getColorTexture(void) const;
    GLuint getDepthTexture(void) const;

    int getWidth(void) const;
    int getHeight(void) const;
private:
    GLuint f, c, d;
    int width, height;

    static std::vector<GLuint> f_stack;

    friend FrameBufferDrawer;
};

//--------------------------------------------------------------------------
class FrameBufferGetter {
public:
    static const FrameBuffer& get(int w, int h, bool isClear);
    static void unget(void);
    static void clear(void);
private:
    static std::vector<std::shared_ptr<FrameBuffer>> f_stack;
    static int pos;
```

```
    static bool isMustClear;
};

//--------------------------------------------------------------------------------
class FrameBufferMerger
{
public:
    static void merge(const FrameBuffer& f1, const FrameBuffer& f2);
private:
    GLuint program;
    GLuint c1ID, d1ID, c2ID, d2ID;

    FrameBufferMerger();
};

//--------------------------------------------------------------------------------
class FrameBufferDrawer
{
public:
    static void draw(const FrameBuffer& f1);
private:
    GLuint program;
    GLuint cID, dID;

    FrameBufferDrawer();
};

//--------------------------------------------------------------------------------
class ScreenFiller
{
public:
    static void fill(void);
private:
    GLuint quad_vertexbuffer;

    ScreenFiller();
};

//--------------------------------------------------------------------------------
class PolygonFramebufferDrawer
{
public:
    static void draw(const FrameBuffer& f1, const std::vector<Fragment>&
    ↪  fragments);
private:
    GLuint program;
    GLuint cID, dID;

    PolygonFramebufferDrawer();
};
```

FILE **opengl_common.h**

```
#pragma once

#include <vector>
#include <stack>

#include <GL/glew.h>
#include <glm/glm.hpp>
```

```cpp
#include <spob/spob.h>

#include <prtl_vis/plane.h>
#include <prtl_vis/fragment.h>
#include <prtl_vis/scene_reader.h>

//------------------------------------------------------------------------------
class SceneDrawer
{
public:
    SceneDrawer(const scene::Scene& scene, glm::vec3& cam_rotate_around,
    ↪  glm::vec3& cam_spheric_pos, int maxDepth);

    void setCam(glm::vec3& cam_rotate_around, glm::vec3& cam_spheric_pos);

    int drawAll(int width, int height);

    void setMaxDepth(int maxDepth) { depthMax = maxDepth; }
    int getMaxDepth(void) const { return depthMax; }

    int getCurrentFrame(void) const { return frame+1; }
    int getMaxFrame(void) const { return frame_max; }

    void turnLight(void) { isDrawLight = !isDrawLight; }

    SceneDrawer& operator++(void);
    SceneDrawer& operator--(void);
private:
    struct PortalToDraw
    {
        std::vector<glm::vec4> polygon;
        std::vector<Fragment> fragments;
        glm::mat4 teleport;
        Plane plane;
        bool isInvert;
        bool isTeleportInvert;
        glm::vec3 color;
    };

    struct ColoredPolygonToDraw {
        std::vector<Fragment> fragments;
        glm::vec3 color;
    };

    struct TexturedPolygonToDraw {
        std::vector<TexFragment> fragments;
        GLuint texture;
    };

    struct Frame
    {
        std::vector<scene::Luminary> luminaries;
        std::vector<GLuint> textures;
        std::vector<unsigned char*> texture_data;
        std::vector<ColoredPolygonToDraw> colored_polygons;
        std::vector<TexturedPolygonToDraw> textured_polygons;
        std::vector<PortalToDraw> portals;
    };

```

```
    static std::pair<PortalToDraw, PortalToDraw> makeDrawPortal(
        const std::vector<spob::vec2>& polygon,
        const spob::space3& crd1, const spob::space3& crd2,
        const spob::vec3& clr1, const spob::vec3& clr2
    );
    void drawScene(int depth);
    void drawPortal(const PortalToDraw& portal, int depth);
    void enableLight(void);
    void disableLight(void);

    std::vector<Frame> frames;
    int depthMax;
    int w, h;
    int frame;
    int frame_max;
    int drawSceneCount;
    std::stack<int> currentDrawPortal;
    bool clockWiseInvert;
    bool isDrawLight;
    std::stack<glm::mat4> currentTeleportMatrix;
    std::stack<std::vector<std::vector<glm::vec4>>> projectedPortalView;

    glm::vec3 cam_1, cam_2;
};

//--------------------------------------------------------------------------
glm::mat4 getFromMatrix(const spob::crd3& crd);
glm::mat4 getToMatrix(const spob::crd3& crd);

glm::vec4 spob2glm(const spob::vec2& vec);
glm::vec4 spob2glm(const spob::vec3& vec);
std::vector<glm::vec4> spob2glm(const std::vector<spob::vec3>& mas);
std::vector<glm::vec4> spob2glm(const std::vector<spob::vec2>& mas);
std::vector<glm::vec4> spob2glm(const std::vector<spob::vec2>& mas, const
↪  spob::plane3& plane);

glm::vec3 spheric2cartesian(glm::vec3 cartesian);
glm::vec3 cartesian2spheric(glm::vec3 spheric);

std::vector<glm::vec4> projectPolygonToScreen(const std::vector<glm::vec4>&
↪  polygon);
std::vector<std::vector<glm::vec4>> intersect(const
↪  std::vector<std::vector<glm::vec4>>& a, const std::vector<glm::vec4>& b);

template<class T>
bool isPolygonOrientedClockwise(const std::vector<T>& polygon);
template<class T>
std::vector<T> orientPolygonClockwise(const std::vector<T>& polygon);

//--------------------------------------------------------------------------
template<class T>
bool isPolygonOrientedClockwise(const std::vector<T>& polygon) {
    double sum = 0;
    for (int i = 0; i < polygon.size() - 1; i++)
        sum += (polygon[i + 1].x - polygon[i].x)*(polygon[i + 1].y +
        ↪  polygon[i].y);
    sum += (polygon[0].x - polygon[polygon.size() - 1].x)*(polygon[0].y +
    ↪  polygon[polygon.size() - 1].y);
    return sum > 0;
```

```
}

//---------------------------------------------------------------------------
template<class T>
std::vector<T> orientPolygonClockwise(const std::vector<T>& polygon) {
    if (isPolygonOrientedClockwise(polygon))
        return polygon;
    else
        return std::vector<T>(polygon.rbegin(), polygon.rend());
}
```

FILE **plane.h**

```
#pragma once

#include <vector>
#include <glm/glm.hpp>

//---------------------------------------------------------------------------
struct Plane : public glm::vec4 {
    Plane() : glm::vec4() {}
    Plane(const glm::vec4& v) : glm::vec4(v) {}
    void invert(void);
};

//---------------------------------------------------------------------------
class ClipPlane {
public:
    static void activate(const Plane& p);
    static void disable(void);
    static Plane getCurrentPlane(void);
private:
    static std::vector<Plane> p_stack;
};

//---------------------------------------------------------------------------
glm::vec4 getClipPlaneEquation(void);
bool isPointBehindPlane(const Plane& plane, const glm::vec4& point);
bool isPolygonBehindPlane(const Plane& plane, const std::vector<glm::vec4>&
↪  polygon);
```

FILE **scene_reader.h**

```
#pragma once

#include <string>
#include <vector>
#include <spob/spob.h>
#include <json.hpp>

namespace scene
{
    using json = nlohmann::json;

    struct Scene;
    struct Frame;
    struct TexturedPolygon;
    struct ColoredPolygon;
    struct Portal;

```

```
    struct Portal
    {
        spob::space3 crd1, crd2;
        std::vector<spob::vec2> polygon;
        spob::vec3 color1, color2;
    };

    struct ColoredPolygon
    {
        spob::plane3 crd;
        std::vector<spob::vec2> polygon;
        spob::vec3 color;
    };

    struct TexturedPolygon
    {
        spob::plane3 crd;
        std::vector<spob::vec2> polygon;
        std::vector<spob::vec2> tex_coords;
        int texture_id;
    };

    struct Texture
    {
        std::string filename;
        int id;
    };

    struct Luminary
    {
        spob::vec3 pos;
        spob::vec3 color;
    };

    struct Frame
    {
        std::vector<Luminary> luminaries;
        std::vector<Texture> textures;
        std::vector<TexturedPolygon> textured_polygons;
        std::vector<ColoredPolygon> colored_polygons;
        std::vector<Portal> portals;
    };

    struct Scene
    {
        spob::vec3 cam_rotate_around, cam_spheric_pos;
        std::vector<Frame> frames;
    };

    Scene parseScene(const json& obj);
    Frame parseFrame(const json& obj);
    Luminary parseLuminary(const json& obj);
    Texture parseTexture(const json& obj);
    TexturedPolygon parseTexturedPolygon(const json& obj);
    ColoredPolygon parseColoredPolygon(const json& obj);
    Portal parsePortal(const json& obj);
    spob::crd3 parseCrd3(const json& obj);
    spob::vec3 parseVec3(const json& obj);
    spob::vec2 parseVec2(const json& obj);
```

```
    json unparse(const Scene& scene);
    json unparse(const Frame& frame);
    json unparse(const Luminary& luminary);
    json unparse(const Texture& texture);
    json unparse(const TexturedPolygon& textured_polygon);
    json unparse(const ColoredPolygon& colored_polygon);
    json unparse(const Portal& portal);
    json unparse(const spob::crd3& crd);
    json unparse(const spob::vec3& vec);
    json unparse(const spob::vec2& vec);
};
```

FILE **shader.h**

```
#pragma once

#include <GL/glew.h>

GLuint LoadShaders(const char * vertex_file_path,const char *
↪  fragment_file_path);
```

5.2 Исходные файлы

FILE **main.cpp**

```
#include <fstream>
#include <iostream>
#include <vector>
#include <iomanip>
#include <sstream>
#include <algorithm>
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <spob/spob.h>
#include <array>

#include <json.hpp>

#include <prtl_vis/scene_reader.h>
#include <prtl_vis/opengl_common.h>
#include <prtl_vis/plane.h>
#include <prtl_vis/shader.h>
#include <prtl_vis/framebuffer.h>

SceneDrawer* sceneDrawer;

//------------------------------------------------------------------------------
int depthMax = 3;
double pi = _SPOB_PI;
int w = 800, h = 600;

glm::vec3 cam_spheric_pos;
glm::vec3 cam_rotate_around;

int fps = 0, drawSceneCount1 = 0;
int drawTime = 0, drawCount = 0, drawSceneCount = 0;
```

```
bool drawFps = true;
bool drawSceneDrawed = false;
bool drawCamPos = false;
bool drawDepth = false;
bool drawFrame = true;

void update_cam();

//-----------------------------------------------------------------------------
void printText(int x, int y, const std::string& str) {
    int c = std::count(str.begin(), str.end(), '\n');

    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, w, 0, h);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    //glColor3f(0, 0, 0);
    glColor3f(1, 1, 1);
    c--; glRasterPos2i(x, y + 15 * c);
    for (const auto& i : str) {
        if (i == '\n') {
            c--; glRasterPos2f(x, y + 15 * c);
        }
        else
            glutBitmapCharacter(GLUT_BITMAP_9_BY_15, i);
    }

    update_cam();
}

//-----------------------------------------------------------------------------
void writeFps(int value) {
    if (drawCount != 0) {
        fps = 1000.0 / (drawTime / drawCount);
        drawSceneCount1 = drawSceneCount/drawCount;
        drawTime = 0;
        drawCount = 0;
        drawSceneCount = 0;
    }
    glutTimerFunc(1000, writeFps, 100);
}

//-----------------------------------------------------------------------------
void display() {
    int timeSinceStart = glutGet(GLUT_ELAPSED_TIME);
    glClearColor(0.6, 0.6, 0.3, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT);

    drawSceneCount += sceneDrawer->drawAll(w, h);

    std::stringstream sout;

    if (drawFps) {
        sout << "FPS: " << fps << std::endl;
    }
    if (drawSceneDrawed) {
```

```
        sout << "Scene drawed: " << drawSceneCount1 << std::endl;
    }
    if (drawCamPos) {
        sout << "Cam rotate point: (" << std::fixed << std::setprecision(2)
            << std::setw(6) << cam_rotate_around.x << ", "
            << std::setw(6) << cam_rotate_around.y << ", "
            << std::setw(6) << cam_rotate_around.z << ")" << std::endl;
        sout << "Cam spheric pos:  (" << std::fixed << std::setprecision(2)
            << std::setw(6) << cam_spheric_pos.x << ", "
            << std::setw(6) << cam_spheric_pos.y << ", "
            << std::setw(6) << cam_spheric_pos.z << ")" << std::endl;
    }
    if (drawDepth) {
        sout << "Depth: " << sceneDrawer->getMaxDepth() << std::endl;
    }
    if (drawFrame) {
        sout << "Frame: " << sceneDrawer->getCurrentFrame() << "/" <<
        ↪  sceneDrawer->getMaxFrame() << std::endl;
    }

    if (!sout.str().empty())
        printText(5, 5, sout.str());

    glutSwapBuffers();

    drawTime += glutGet(GLUT_ELAPSED_TIME) - timeSinceStart;
    drawCount++;
}

//--------------------------------------------------------------------------------
void update_cam(void) {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(90.0, double(w)/h, 0.1, 1000.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glm::vec3 pos1 = cam_rotate_around + spheric2cartesian(cam_spheric_pos);
    gluLookAt(pos1.x, pos1.y, pos1.z,
              cam_rotate_around.x, cam_rotate_around.y, cam_rotate_around.z,
              0, 0, 1);
}

//--------------------------------------------------------------------------------
void reshape(int w1, int h1) {
    w = w1; h = h1;
    FrameBufferGetter::clear();
    glViewport(0, 0, w, h);
    update_cam();
    glutPostRedisplay();
}

//--------------------------------------------------------------------------------
int r_moving, r_startx, r_starty;
int l_moving, l_startx, l_starty;
int m_moving, m_startx, m_starty;
void mouse(int button, int state, int x, int y) {
    if (button == GLUT_LEFT_BUTTON) {
        if (state == GLUT_DOWN) {
```

```
            l_moving = 1;
            l_startx = x;
            l_starty = y;
        }
        if (state == GLUT_UP) {
            l_moving = 0;
        }
    }
    /*if (button == GLUT_RIGHT_BUTTON) {
        if (state == GLUT_DOWN) {
            r_moving = 1;
            r_startx = x;
            r_starty = y;
        }
        if (state == GLUT_UP) {
            r_moving = 0;
        }
    }
    if (button == GLUT_MIDDLE_BUTTON) {
        if (state == GLUT_DOWN) {
            m_moving = 1;
            m_startx = x;
            m_starty = y;
        }
        if (state == GLUT_UP) {
            m_moving = 0;
        }
    }*/
}

//-------------------------------------------------------------------------------
void motion(int x, int y) {
    if (l_moving) {
        cam_spheric_pos.x = glm::radians(glm::degrees(cam_spheric_pos.x) -
        ↪  0.5*(x - l_startx));
        cam_spheric_pos.y = glm::radians(glm::degrees(cam_spheric_pos.y) -
        ↪  0.5*(y - l_starty));
        l_startx = x;
        l_starty = y;

        if (cam_spheric_pos.y < 0.01) cam_spheric_pos.y = 0.01;
        if (cam_spheric_pos.y > pi-0.01) cam_spheric_pos.y = pi-0.01;

        update_cam();
        glutPostRedisplay();
    }
    /*if (r_moving) {
        cam_rotate_around.x -= 0.01*(x-r_startx);
        cam_rotate_around.y -= 0.01*(y-r_starty);
        r_startx = x;
        r_starty = y;
        update_cam();
        glutPostRedisplay();
    }
    if (m_moving) {
        cam_rotate_around.z += 0.01*(y-m_starty);
        m_starty = y;
        update_cam();
        glutPostRedisplay();
```

```
    }*/
}

//--------------------------------------------------------------------------------
void wheel(int button, int dir, int x, int y) {
    if (dir < 0) cam_spheric_pos.z += 0.1;
    else cam_spheric_pos.z -= 0.1;

    update_cam();

    glutPostRedisplay();
}

//--------------------------------------------------------------------------------
void keyboard(unsigned char key, int x, int y) {
    /*if (key == 'a') cam_spheric_pos.x =
    ↪  glm::radians(glm::degrees(cam_spheric_pos.x) + 3.0);
    if (key == 'o') cam_spheric_pos.x =
↪  glm::radians(glm::degrees(cam_spheric_pos.x) - 3.0);

    if (key == 'e') cam_spheric_pos.y =
↪  glm::radians(glm::degrees(cam_spheric_pos.y) + 3.0);
    if (key == 'u') cam_spheric_pos.y =
↪  glm::radians(glm::degrees(cam_spheric_pos.y) - 3.0);
    if (cam_spheric_pos.y < 0.01) cam_spheric_pos.y = 0.01;
    if (cam_spheric_pos.y > pi - 0.01) cam_spheric_pos.y = pi - 0.01;*/

    if (key == '{') wheel(0, 1, 0, 0);
    if (key == '}') wheel(0, -1, 0, 0);

    if (key == '+' || key == '=') ++(*sceneDrawer);
    if (key == '-') --(*sceneDrawer);

    if (key == 'h') sceneDrawer->setCam(cam_rotate_around, cam_spheric_pos);

    if (key == 'l') sceneDrawer->turnLight();

    update_cam();

    glutPostRedisplay();
}

//--------------------------------------------------------------------------------
void init() {
}

//--------------------------------------------------------------------------------
void menu(int num) {
    switch (num) {
        case 101: sceneDrawer->setMaxDepth(01); break;
        case 102: sceneDrawer->setMaxDepth(02); break;
        case 103: sceneDrawer->setMaxDepth(03); break;
        case 104: sceneDrawer->setMaxDepth(04); break;
        case 105: sceneDrawer->setMaxDepth(05); break;
        case 106: sceneDrawer->setMaxDepth(06); break;
        case 110: sceneDrawer->setMaxDepth(10); break;
        case 115: sceneDrawer->setMaxDepth(15); break;
        case 120: sceneDrawer->setMaxDepth(20); break;
        case 130: sceneDrawer->setMaxDepth(30); break;
```

```
        case 199: sceneDrawer->setMaxDepth(99); break;

        case 200: drawFps = !drawFps;       break;
        case 201: drawCamPos = !drawCamPos; break;
        case 202: drawDepth = !drawDepth;   break;
        case 203: drawFrame = !drawFrame;   break;
        case 204: drawSceneDrawed = !drawSceneDrawed; break;

        case 0: keyboard('h', 0, 0); break;
        case 1: keyboard('+', 0, 0); break;
        case 2: keyboard('-', 0, 0); break;
        case 3: keyboard('{', 0, 0); break;
        case 4: keyboard('}', 0, 0); break;
        case 5: keyboard('l', 0, 0); break;
    }
    glutPostRedisplay();
}

//------------------------------------------------------------------------------
void createMenu(void) {
    int depthMenu = glutCreateMenu(menu);
    glutAddMenuEntry("1", 101);
    glutAddMenuEntry("2", 102);
    glutAddMenuEntry("3", 103);
    glutAddMenuEntry("4", 104);
    glutAddMenuEntry("5", 105);
    glutAddMenuEntry("6", 106);
    glutAddMenuEntry("10", 110);
    glutAddMenuEntry("15", 115);
    glutAddMenuEntry("20", 120);
    glutAddMenuEntry("30", 130);
    glutAddMenuEntry("99 (Not recommended!)", 199);

    int debugMenu = glutCreateMenu(menu);
    glutAddMenuEntry("FPS", 200);
    glutAddMenuEntry("Cam position", 201);
    glutAddMenuEntry("Depth", 202);
    glutAddMenuEntry("Frame", 203);
    glutAddMenuEntry("Scene drawed count", 204);

    int mainMenu = glutCreateMenu(menu);
    glutAddMenuEntry("Move cam to start position  h", 0);
    glutAddMenuEntry("Next frame                  +", 1);
    glutAddMenuEntry("Previous frame              -", 2);
    glutAddMenuEntry("Zoom in                     {", 3);
    glutAddMenuEntry("Zoom out                    }", 4);
    glutAddMenuEntry("Turn light off/on           l", 5);
    glutAddSubMenu("Depth", depthMenu);
    glutAddSubMenu("Show debug information", debugMenu);

    glutAttachMenu(GLUT_RIGHT_BUTTON);
}

//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------

int main(int argc, char** argv) {
    std::string filename = "scene.json";
```

```
322     if (argc > 1)
323         filename = std::string(argv[1]);
324 
325     scene::json js;
326     std::ifstream fin(filename);
327     fin >> js;
328     fin.close();
329 
330     glutInit(&argc, argv);
331     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH | GLUT_STENCIL);
332     glutInitWindowPosition(80, 80);
333     glutInitWindowSize(w, h);
334     glutCreateWindow("Portal viewer");
335 
336     // Initialize GLEW
337     glewExperimental = true;
338     if (glewInit() != GLEW_OK) {
339         fprintf(stderr, "Failed to initialize GLEW\n");
340         getchar();
341         return -1;
342     }
343 
344     glutDisplayFunc(display);
345     glutReshapeFunc(reshape);
346 
347     glutMouseFunc(mouse);
348     glutMotionFunc(motion);
349     glutKeyboardFunc(keyboard);
350     glutMouseWheelFunc(wheel);
351     glutTimerFunc(1000, writeFps, 100);
352 
353     glEnable(GL_DEPTH_TEST);
354     glColor3f(1.0f, 1.0f, 1.0f);
355 
356     // И почему это ломает цвета, когда нет текстур, а?
357     //glEnable(GL_TEXTURE_2D);
358     //glBindTexture(GL_TEXTURE_2D, 0);
359 
360     init();
361     createMenu();
362     sceneDrawer = new SceneDrawer(scene::parseScene(js), cam_rotate_around,
    ↪  cam_spheric_pos, 6);
363     glutMainLoop();
364 }
```

FILE **fragment.cpp**

```
1  #include <GL/glew.h>
2 
3  #include <prtl_vis/fragment.h>
4 
5  std::vector<Fragment> Fragmentator::fragments;
6  std::vector<TexFragment> Fragmentator::texFragments;
7 
8  //--------------------------------------------------------------------------
9  void __stdcall  Fragmentator::tessBegin1(GLenum which) {
10     fragments.push_back({int(which), {}});
11 }
12 
13 //--------------------------------------------------------------------------
```

```
void __stdcall Fragmentator::tessEnd1() {
}

//--------------------------------------------------------------------------------
void __stdcall Fragmentator::tessVertex1(const GLvoid *data) {
    const GLdouble* ptr = (GLdouble*)data;
    fragments.back().vertices.emplace_back(ptr[0], ptr[1], ptr[2]);
}

//--------------------------------------------------------------------------------
void __stdcall Fragmentator::tessError1(GLenum errorCode) {
    throw std::exception();
}

//--------------------------------------------------------------------------------
void __stdcall  Fragmentator::tessBegin2(GLenum which) {
    texFragments.push_back({int(which), {}, {}});
}

//--------------------------------------------------------------------------------
void __stdcall Fragmentator::tessEnd2() {
}

//--------------------------------------------------------------------------------
void __stdcall Fragmentator::tessVertex2(const GLvoid *data) {
    const GLdouble* ptr = (GLdouble*)data;
    texFragments.back().vertices.emplace_back(ptr[0], ptr[1], ptr[2]);
    texFragments.back().tex_coords.emplace_back(ptr[3], ptr[4]);
}

//--------------------------------------------------------------------------------
void __stdcall Fragmentator::tessError2(GLenum errorCode) {
    throw std::exception();
}

//--------------------------------------------------------------------------------
std::vector<Fragment> Fragmentator::fragmentize(const std::vector<glm::vec4>&
↪   polygon) {
    // http://www.songho.ca/opengl/gl_tessellation.html
    fragments.clear();
    GLuint id = glGenLists(1);
    if (!id) throw std::exception();

    GLUtesselator *tess = gluNewTess();
    if (!tess) throw std::exception();

    std::vector<glm::dvec4> p;
    for (auto& i : polygon)
        p.push_back(i);

    gluTessCallback(tess, GLU_TESS_BEGIN, (void (__stdcall *)())tessBegin1);
    gluTessCallback(tess, GLU_TESS_END, (void (__stdcall *)())tessEnd1);
    gluTessCallback(tess, GLU_TESS_ERROR, (void(__stdcall*)(void))tessError1);
    gluTessCallback(tess, GLU_TESS_VERTEX, (void (__stdcall *)())tessVertex1);

    glNewList(id, GL_COMPILE);
    glColor3f(1, 1, 1);
    gluTessBeginPolygon(tess, 0);
    gluTessBeginContour(tess);
```

```
    for (auto& i : p) {
        gluTessVertex(tess, &i[0], &i[0]);
    }
    gluTessEndContour(tess);
    gluTessEndPolygon(tess);
    glEndList();

    gluDeleteTess(tess);

    return fragments;
}

//-----------------------------------------------------------------------------
std::vector<TexFragment> Fragmentator::fragmentize(const std::vector<glm::vec4>&
 ↪  polygon, const std::vector<glm::vec4>& tex_coords) {
    texFragments.clear();
    GLuint id = glGenLists(1);
    if (!id) throw std::exception();

    GLUtesselator *tess = gluNewTess();
    if (!tess) throw std::exception();

    std::vector<std::vector<GLdouble>> temp;
    for (int i = 0; i < polygon.size(); i++) {
        auto p = polygon[i];
        auto t = tex_coords[i];
        temp.push_back({p.x, p.y, p.z, t.x, t.y});
    }

    gluTessCallback(tess, GLU_TESS_BEGIN, (void(__stdcall *)())tessBegin2);
    gluTessCallback(tess, GLU_TESS_END, (void(__stdcall *)())tessEnd2);
    gluTessCallback(tess, GLU_TESS_ERROR, (void(__stdcall*)(void))tessError2);
    gluTessCallback(tess, GLU_TESS_VERTEX, (void(__stdcall *)())tessVertex2);

    glNewList(id, GL_COMPILE);
    glColor3f(1, 1, 1);
    gluTessBeginPolygon(tess, 0);
    gluTessBeginContour(tess);
    for (auto& i : temp) {
        gluTessVertex(tess, &i[0], &i[0]);
    }
    gluTessEndContour(tess);
    gluTessEndPolygon(tess);
    glEndList();

    gluDeleteTess(tess);

    return texFragments;
}

//-----------------------------------------------------------------------------
void drawFragment(const Fragment& f) {
    glBegin(f.begin);
    for (const auto& i : f.vertices)
        glVertex3f(i.x, i.y, i.z);
    glEnd();
}

//-----------------------------------------------------------------------------
```

```
void drawFragments(const std::vector<Fragment>& f) {
    for (const auto& i : f)
        drawFragment(i);
}

//--------------------------------------------------------------------------
void drawFragment(const TexFragment& f) {
    glBegin(f.begin);
    for (int i = 0; i < f.vertices.size(); i++) {
        auto p = f.vertices[i];
        auto t = f.tex_coords[i];
        glTexCoord2f(t.x, t.y); glVertex3f(p.x, p.y, p.z);
    }
    glEnd();
}

//--------------------------------------------------------------------------
void drawFragments(const std::vector<TexFragment>& f) {
    for (const auto& i : f)
        drawFragment(i);
}
```

FILE **framebuffer.cpp**

```
#include <memory>

#include <GL/glew.h>

#include <prtl_vis/shader.h>
#include <prtl_vis/framebuffer.h>

std::vector<GLuint> FrameBuffer::f_stack(1, 0);

//--------------------------------------------------------------------------
FrameBuffer::FrameBuffer(int width, int height) : width(width), height(height) {
    glGenFramebuffers(1, &f);
    glBindFramebuffer(GL_FRAMEBUFFER, f);

    glGenTextures(1, &c);
    glBindTexture(GL_TEXTURE_2D, c);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
    ↪  GL_UNSIGNED_BYTE, 0);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

    /*glGenTextures(1, &d);
    glBindTexture(GL_TEXTURE_2D, d);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT24, width, height, 0,
↪   GL_DEPTH_COMPONENT, GL_FLOAT, 0);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

    glFramebufferTexture(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, c, 0);
    glFramebufferTexture(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, d, 0);

    GLenum DrawBuffers[3] = {GL_COLOR_ATTACHMENT0, GL_DEPTH_ATTACHMENT};
```

```
35      glDrawBuffers(3, DrawBuffers);
36
37      if(glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE)
38          throw std::exception();
39
40      glBindFramebuffer(GL_FRAMEBUFFER, 0);*/
41
42      glGenTextures(1, &d);
43      glBindTexture(GL_TEXTURE_2D, d);
44      glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH24_STENCIL8, width, height, 0,
        ↪  GL_DEPTH_STENCIL, GL_UNSIGNED_INT_24_8, 0);
45      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
46      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
47      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
48      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
49      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE,
        ↪  GL_COMPARE_R_TO_TEXTURE);
50      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC, GL_LEQUAL);
51
52      glFramebufferTexture(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, c, 0);
53      glFramebufferTexture(GL_FRAMEBUFFER, GL_DEPTH_STENCIL_ATTACHMENT, d, 0);
54
55      GLenum DrawBuffers[3] = {GL_COLOR_ATTACHMENT0, GL_DEPTH_STENCIL_ATTACHMENT};
56      glDrawBuffers(3, DrawBuffers);
57
58      if(glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE)
59          throw std::exception();
60
61      activate();
62      disable();
63
64      glBindFramebuffer(GL_FRAMEBUFFER, 0);
65  }
66
67  //-------------------------------------------------------------------------
68  FrameBuffer::~FrameBuffer() {
69      glDeleteFramebuffers(1, &f);
70      glDeleteTextures(1, &c);
71      glDeleteTextures(1, &d);
72  }
73
74  //-------------------------------------------------------------------------
75  void FrameBuffer::activate(bool isClear) const {
76      f_stack.push_back(f);
77      // Render to our framebuffer
78      glBindFramebuffer(GL_FRAMEBUFFER, f);
79      glViewport(0, 0, width, height); // Render on the whole framebuffer,
        ↪  complete from the lower left corner to the upper right
80
81      if (isClear) {
82          // Clear the screen
83          //glClearColor(0.3, 0.3, 0.3, 1.0f);
84          glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
85      }
86  }
87
88  //-------------------------------------------------------------------------
89  void FrameBuffer::disable(bool isClear) const {
90      f_stack.pop_back();
```

```
    // Render to the screen
    glBindFramebuffer(GL_FRAMEBUFFER, f_stack.back());
    // Render on the whole framebuffer, complete from the lower left corner to
    ↪  the upper right
    glViewport(0, 0, width, height);
    // Clear the screen

    if (isClear) {
        //glClearColor(0.3, 0.3, 0.3, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    }
}

//--------------------------------------------------------------------------
GLuint FrameBuffer::getFrameBuffer(void) const {
    return f;
}

//--------------------------------------------------------------------------
GLuint FrameBuffer::getColorTexture(void) const {
    return c;
}

//--------------------------------------------------------------------------
GLuint FrameBuffer::getDepthTexture(void) const {
    return d;
}

//--------------------------------------------------------------------------
int FrameBuffer::getWidth(void) const {
    return width;
}

//--------------------------------------------------------------------------
int FrameBuffer::getHeight(void) const {
    return height;
}

//--------------------------------------------------------------------------
//--------------------------------------------------------------------------
//--------------------------------------------------------------------------

//--------------------------------------------------------------------------
std::vector<std::shared_ptr<FrameBuffer>> FrameBufferGetter::f_stack;
int FrameBufferGetter::pos(0);
bool FrameBufferGetter::isMustClear(false);

//--------------------------------------------------------------------------
const FrameBuffer& FrameBufferGetter::get(int w, int h, bool isClear) {
    if (pos == f_stack.size())
        f_stack.emplace_back(new FrameBuffer(w, h));
    const FrameBuffer& result(*f_stack[pos]);
    if (isClear) {
        result.activate();
        result.disable();
    }
    pos++;
    return result;
}
```

```
149
150 //--------------------------------------------------------------------------
151 void FrameBufferGetter::unget(void) {
152     pos--;
153     if (pos == 0 && isMustClear) {
154         clear();
155     }
156 }
157
158 //--------------------------------------------------------------------------
159 void FrameBufferGetter::clear(void) {
160     if (pos == 0) {
161         f_stack.clear();
162         pos = 0;
163         isMustClear = false;
164     } else {
165         isMustClear = true;
166     }
167 }
168
169 //--------------------------------------------------------------------------
170 //--------------------------------------------------------------------------
171 //--------------------------------------------------------------------------
172
173 //--------------------------------------------------------------------------
174 FrameBufferMerger::FrameBufferMerger() {
175     program = LoadShaders("merge.vertex.glsl", "merge.fragment.glsl" );
176     c1ID = glGetUniformLocation(program, "Color1");
177     d1ID = glGetUniformLocation(program, "Depth1");
178     c2ID = glGetUniformLocation(program, "Color2");
179     d2ID = glGetUniformLocation(program, "Depth2");
180 }
181
182 //--------------------------------------------------------------------------
183 void FrameBufferMerger::merge(const FrameBuffer& f1, const FrameBuffer& f2) {
184     static FrameBufferMerger merger;
185     glUseProgram(merger.program);
186         glActiveTexture(GL_TEXTURE0);
187         glBindTexture(GL_TEXTURE_2D, f1.getColorTexture());
188         glActiveTexture(GL_TEXTURE1);
189         glBindTexture(GL_TEXTURE_2D, f1.getDepthTexture());
190
191         glActiveTexture(GL_TEXTURE2);
192         glBindTexture(GL_TEXTURE_2D, f2.getColorTexture());
193         glActiveTexture(GL_TEXTURE3);
194         glBindTexture(GL_TEXTURE_2D, f2.getDepthTexture());
195
196         glUniform1i(merger.c1ID, 0);
197         glUniform1i(merger.d1ID, 1);
198         glUniform1i(merger.c2ID, 2);
199         glUniform1i(merger.d2ID, 3);
200
201         ScreenFiller::fill();
202     glUseProgram(0);
203     glActiveTexture(GL_TEXTURE0);
204 }
205
206 //--------------------------------------------------------------------------
207 //--------------------------------------------------------------------------
```

```
208 //----------------------------------------------------------------------------
209
210 //----------------------------------------------------------------------------
211 FrameBufferDrawer::FrameBufferDrawer() {
212     program = LoadShaders(”draw.vertex.glsl”, ”draw.fragment.glsl” );
213     cID = glGetUniformLocation(program, ”Color”);
214     dID = glGetUniformLocation(program, ”Depth”);
215 }
216
217 //----------------------------------------------------------------------------
218 void FrameBufferDrawer::draw(const FrameBuffer& f1) {
219     static FrameBufferDrawer drawer;
220     /*glUseProgram(drawer.program);
221         glActiveTexture(GL_TEXTURE0);
222         glBindTexture(GL_TEXTURE_2D, f1.getColorTexture());
223         glActiveTexture(GL_TEXTURE1);
224         glBindTexture(GL_TEXTURE_2D, f1.getDepthTexture());
225
226         glUniform1i(drawer.cID, 0);
227         glUniform1i(drawer.dID, 1);
228
229         ScreenFiller::fill();
230     glUseProgram(0);*/
231     glBindFramebuffer(GL_READ_FRAMEBUFFER, f1.getFrameBuffer());
232     glBindFramebuffer(GL_DRAW_FRAMEBUFFER, FrameBuffer::f_stack.back());
233     glBlitFramebuffer(0, 0, f1.getWidth(), f1.getHeight(),
234                       0, 0, f1.getWidth(), f1.getHeight(),
235                       GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT, GL_NEAREST);
236 }
237
238 //----------------------------------------------------------------------------
239 //----------------------------------------------------------------------------
240 //----------------------------------------------------------------------------
241
242 //----------------------------------------------------------------------------
243 ScreenFiller::ScreenFiller() {
244     static const GLfloat g_quad_vertex_buffer_data[] = {
245         -1.0f, -1.0f, 0.0f,
246          1.0f, -1.0f, 0.0f,
247         -1.0f,  1.0f, 0.0f,
248         -1.0f,  1.0f, 0.0f,
249          1.0f, -1.0f, 0.0f,
250          1.0f,  1.0f, 0.0f,
251     };
252
253     glGenBuffers(1, &quad_vertexbuffer);
254     glBindBuffer(GL_ARRAY_BUFFER, quad_vertexbuffer);
255     glBufferData(GL_ARRAY_BUFFER, sizeof(g_quad_vertex_buffer_data),
     ↪  g_quad_vertex_buffer_data, GL_STATIC_DRAW);
256 }
257
258 //----------------------------------------------------------------------------
259 void ScreenFiller::fill(void) {
260     static ScreenFiller filler;
261
262     glEnableVertexAttribArray(0);
263     glBindBuffer(GL_ARRAY_BUFFER, filler.quad_vertexbuffer);
264     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, (void*)0);
265     glDrawArrays(GL_TRIANGLES, 0, 6);
```

```
    glDisableVertexAttribArray(0);
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
PolygonFramebufferDrawer::PolygonFramebufferDrawer() {
    program = LoadShaders("drawpoly.vertex.glsl", "drawpoly.fragment.glsl" );
    cID = glGetUniformLocation(program, "Color");
    dID = glGetUniformLocation(program, "Depth");
}

//-----------------------------------------------------------------------------
void PolygonFramebufferDrawer::draw(const FrameBuffer& f1, const
↪   std::vector<Fragment>& fragments) {
    static PolygonFramebufferDrawer drawer;
    glUseProgram(drawer.program);
        glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_2D, f1.getColorTexture());
        glActiveTexture(GL_TEXTURE1);
        glBindTexture(GL_TEXTURE_2D, f1.getDepthTexture());

        glUniform1i(drawer.cID, 0);
        glUniform1i(drawer.dID, 1);

        /*glBegin(GL_POLYGON);
        for (auto& i : poly)
            glVertex3f(i.x, i.y, i.z);
        glEnd();*/
        drawFragments(fragments);
    glUseProgram(0);
    glActiveTexture(GL_TEXTURE0);
}
```

FILE **opengl_common.cpp**

```
#define STB_IMAGE_IMPLEMENTATION
#include <stb_image.h>

#include <array>
#include <glm/glm.hpp>
#include <glm/gtx/transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include <prtl_vis/plane.h>
#include <prtl_vis/framebuffer.h>
#include <prtl_vis/opengl_common.h>

#include <clipper.hpp>

//-----------------------------------------------------------------------------
SceneDrawer::SceneDrawer(const scene::Scene& scene, glm::vec3&
↪   cam_rotate_around, glm::vec3& cam_spheric_pos, int maxDepth) :
↪   depthMax(maxDepth), frame(0), isDrawLight(false) {
    cam_1 = cam_rotate_around = spob2glm(scene.cam_rotate_around);
    cam_2 = cam_spheric_pos = spob2glm(scene.cam_spheric_pos);
    for (auto& i : scene.frames) {
        frames.emplace_back();
        Frame& f = frames.back();
```

```
        for (auto& j : i.portals) {
            auto result = makeDrawPortal(orientPolygonClockwise(j.polygon),
            ↪  j.crd1, j.crd2, j.color1, j.color2);
            f.portals.push_back(result.first);
            f.portals.push_back(result.second);
        }
        for (auto& j : i.colored_polygons) {
            f.colored_polygons.push_back({
                Fragmentator::fragmentize(spob2glm(orientPolygonClockwise(j.poly
                ↪  gon),
                ↪  j.crd)),
                spob2glm(j.color)
            });
        }

        // Считываем текстуры
        if (i.textures.size() != 0) {
            int texN = i.textures.size();
            f.textures.resize(texN, 0);
            f.texture_data.resize(texN, nullptr);
            glGenTextures(texN, &f.textures[0]);
            for (int j = 0; j < i.textures.size(); j++) {
                int width, height, n;
                f.texture_data[j] = stbi_load(i.textures[j].filename.c_str(),
                ↪  &width, &height, &n, 3);
                glBindTexture(GL_TEXTURE_2D, f.textures[j]);
                glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
                glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                ↪  GL_LINEAR_MIPMAP_LINEAR);
                gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, width, height, GL_RGB,
                ↪  GL_UNSIGNED_BYTE, f.texture_data[j]);
                glBindTexture(GL_TEXTURE_2D, 0);
            }
        }

        // Считываем текстурированные полигоны
        for (auto& j : i.textured_polygons) {
            auto poly = spob2glm(j.polygon, j.crd);
            auto tex_coords = spob2glm(j.tex_coords);
            if (!isPolygonOrientedClockwise(poly)) {
                poly = std::vector<glm::vec4>(poly.rbegin(), poly.rend());
                tex_coords = std::vector<glm::vec4>(tex_coords.rbegin(),
                ↪  tex_coords.rend());
            }
            f.textured_polygons.push_back({
                Fragmentator::fragmentize(poly, tex_coords),
                f.textures[j.texture_id]
            });
        }

        // Считываем источники освещения
        f.luminaries = i.luminaries;
    }
    frame_max = frames.size();
}

//----------------------------------------------------------------------------
void SceneDrawer::setCam(glm::vec3& cam_rotate_around, glm::vec3&
↪  cam_spheric_pos) {
```

```
72      cam_rotate_around = cam_1;
73      cam_spheric_pos = cam_2;
74  }
75
76  //--------------------------------------------------------------------------------
77  int SceneDrawer::drawAll(int width, int height) {
78      const auto& portals = frames[frame].portals;
79
80      w = width; h = height;
81
82      projectedPortalView.push({ { {0, 0, 0, 0}, {0, h, 0, 0}, {w, h, 0, 0}, {w,
        ↪  0, 0, 0} } });
83      drawSceneCount = 0;
84      clockWiseInvert = false;
85      const FrameBuffer& f = FrameBufferGetter::get(w, h, true);
86      f.activate();
87      drawScene(1);
88      f.disable();
89      FrameBufferDrawer::draw(f);
90      FrameBufferGetter::unget();
91      projectedPortalView.pop();
92
93      return drawSceneCount;
94  }
95
96  //--------------------------------------------------------------------------------
97  void SceneDrawer::drawPortal(const PortalToDraw& portal, int depth) {
98      if (depth > depthMax) return;
99
100     const auto& portals = frames[frame].portals;
101
102     // Проверка на то, находится ли полигон внутри рисуемой полуплоскости
103     bool isBehindPlane = depth == 1;
104     if (depth != 1) {
105         auto clipPlane = ClipPlane::getCurrentPlane() *
            ↪  currentTeleportMatrix.top();
106         isBehindPlane = isPolygonBehindPlane(clipPlane, portal.polygon);
107     }
108     if (isBehindPlane) {
109         auto projected = projectPolygonToScreen(portal.polygon);
110
111         auto intersected = intersect(projectedPortalView.top(), projected);
112         bool isVisibleOnScreen = intersected.size() != 0;
113         if (!isVisibleOnScreen) return;
114
115         projectedPortalView.push(intersected);
116
117         // Определяем, как ориентирован портал. Если по часовой стрелке, то
            ↪  можно рисовать, иначе рисуем обратную сторону портала.
118         bool isInvert = portal.isInvert;
119         if (clockWiseInvert) isInvert = !isInvert;
120         bool isDraw = isPolygonOrientedClockwise(projected);
121         if (!isInvert) isDraw = !isDraw;
122         if (isDraw) {
123             if (portal.isTeleportInvert) clockWiseInvert = !clockWiseInvert;
124
125             // Рисуем портал и сцену с ним
126             const FrameBuffer& f = FrameBufferGetter::get(w, h, true);
127             f.activate();
```

```
128             ClipPlane::activate(portal.plane);
129                 glMatrixMode(GL_MODELVIEW); glPushMatrix();
130                 glMultMatrixf(glm::value_ptr(portal.teleport));
131                 currentTeleportMatrix.push(portal.teleport);
132                     drawScene(depth + 1);
133                 currentTeleportMatrix.pop();
134                 glMatrixMode(GL_MODELVIEW); glPopMatrix();
135 
136             // Нельзя просто отключить плоскость, необходимо вернуть ту матрицу
                ↪  модельно-видового преобразования, которая была при включении
                ↪  этой плоскости.
137             // До того, как был написан код для вовзращения матрицы, этот код
                ↪  был местом серьезного бага
138             if (!currentTeleportMatrix.empty()) {
139                 glPopMatrix();
140                 ClipPlane::disable();
141                 glPushMatrix();
142                 glMultMatrixf(glm::value_ptr(currentTeleportMatrix.top()));
143             } else
144                 ClipPlane::disable();
145             f.disable();
146 
147             PolygonFramebufferDrawer::draw(f, portal.fragments);
148             FrameBufferGetter::unget();
149 
150             if (portal.isTeleportInvert) clockWiseInvert = !clockWiseInvert;
151         } else {
152             // Рисуем обратную сторону портала с указанным цветом
153             glDisable(GL_TEXTURE_2D);
154             enableLight();
155             glColor3f(portal.color.x, portal.color.y, portal.color.z);
156             drawFragments(portal.fragments);
157             disableLight();
158         }
159 
160         projectedPortalView.pop();
161     }
162 }
163 
164 //----------------------------------------------------------------------------
165 void SceneDrawer::enableLight(void) {
166     const auto& luminaries = frames[frame].luminaries;
167     if (isDrawLight) {
168         glEnable(GL_COLOR_MATERIAL);
169         glEnable(GL_NORMALIZE);
170         glEnable(GL_LIGHTING);
171         for (int i = 0; i < std::min<int>(luminaries.size(), 7); i++) {
172             scene::Luminary l = luminaries[i];
173             GLfloat light_diffuse[] = {l.color.x, l.color.y, l.color.z};
174             GLfloat light_position[] = {l.pos.x, l.pos.y, l.pos.z, 1.0};
175             glEnable(GL_LIGHT0 + i);
176             glLightfv(GL_LIGHT0 + i, GL_DIFFUSE, light_diffuse);
177             glLightfv(GL_LIGHT0 + i, GL_POSITION, light_position);
178             glLightf(GL_LIGHT0 + i, GL_CONSTANT_ATTENUATION, 0.0);
179             glLightf(GL_LIGHT0 + i, GL_LINEAR_ATTENUATION, 0.05);
180             glLightf(GL_LIGHT0 + i, GL_QUADRATIC_ATTENUATION, 0);
181         }
182     }
183 }
```

```
184
185 //-----------------------------------------------------------------------------
186 void SceneDrawer::disableLight(void) {
187     const auto& luminaries = frames[frame].luminaries;
188     if (isDrawLight) {
189         for (int i = 0; i < std::min<int>(luminaries.size(), 7); i++) {
190             glDisable(GL_LIGHT0 + i);
191         }
192         glDisable(GL_LIGHTING);
193         glDisable(GL_NORMALIZE);
194         glDisable(GL_COLOR_MATERIAL);
195     }
196 }
197
198 //-----------------------------------------------------------------------------
199 void SceneDrawer::drawScene(int depth) {
200     if (depth > depthMax) return;
201
202     drawSceneCount++;
203
204     const auto& textured_polygons = frames[frame].textured_polygons;
205     const auto& colored_polygons = frames[frame].colored_polygons;
206     const auto& portals = frames[frame].portals;
207
208     //-----------------------------------------------------------------------------
209     // Рисуем все порталы
210     const FrameBuffer& f = FrameBufferGetter::get(w, h, true);
211     const FrameBuffer& f1 = FrameBufferGetter::get(w, h, true);
212
213     for (int i = 0; i < portals.size(); ++i) {
214         bool isDraw = true;
215         if (!currentDrawPortal.empty()) {
216             if (i % 2 == 1)
217                 isDraw = i-1 != currentDrawPortal.top();
218             else
219                 isDraw = i+1 != currentDrawPortal.top();
220         }
221         if (isDraw) {
222             currentDrawPortal.push(i);
223             f1.activate();
224             drawPortal(portals[i], depth);
225             f1.disable(false);
226             currentDrawPortal.pop();
227
228             f.activate(false);
229             FrameBufferMerger::merge(f, f1);
230             f.disable(false);
231         }
232     }
233
234     FrameBufferDrawer::draw(f);
235
236     FrameBufferGetter::unget();
237     FrameBufferGetter::unget();
238
239     enableLight();
240
241     //-----------------------------------------------------------------------------
242     // Рисуем все полигоны
```

```
    glDisable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, 0);
    for (auto& i : colored_polygons) {
        glColor3f(i.color.x, i.color.y, i.color.z);
        drawFragments(i.fragments);
    }
    glEnable(GL_TEXTURE_2D);

    for (auto& i : textured_polygons) {
        glColor3f(1, 1, 1);
        glBindTexture(GL_TEXTURE_2D, i.texture);
        drawFragments(i.fragments);
        glBindTexture(GL_TEXTURE_2D, 0);
    }

    disableLight();
}

//------------------------------------------------------------------------------
SceneDrawer& SceneDrawer::operator++(void) {
    if (frame+1 == frame_max) frame = 0;
    else frame++;
    return *this;
}

//------------------------------------------------------------------------------
SceneDrawer& SceneDrawer::operator--(void) {
    if (frame == 0) frame = frame_max-1;
    else frame--;
    return *this;
}

//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
std::pair<SceneDrawer::PortalToDraw, SceneDrawer::PortalToDraw>
    ↪ SceneDrawer::makeDrawPortal(const std::vector<spob::vec2>& polygon, const
    ↪ spob::space3& crd1, const spob::space3& crd2, const spob::vec3& clr1, const
    ↪ spob::vec3& clr2) {
    PortalToDraw p1, p2;

    p1.teleport = getFromMatrix(crd2) * getToMatrix(crd1);
    p2.teleport = getFromMatrix(crd1) * getToMatrix(crd2);

    for (auto& i : polygon) {
        p2.polygon.push_back(spob2glm(spob::plane3(crd1).from(i)));
        p1.polygon.push_back(spob2glm(spob::plane3(crd2).from(i)));
    }

    p1.fragments = Fragmentator::fragmentize(p1.polygon);
    p2.fragments = Fragmentator::fragmentize(p2.polygon);

    p1.plane.x = crd1.k.x;
    p1.plane.y = crd1.k.y;
    p1.plane.z = crd1.k.z;
    p1.plane.w = -dot(crd1.pos, crd1.k);

```

```
    p2.plane.x = -crd2.k.x;
    p2.plane.y = -crd2.k.y;
    p2.plane.z = -crd2.k.z;
    p2.plane.w = dot(crd2.pos, crd2.k);

    std::swap(p1.plane, p2.plane);
    p1.plane.invert();
    p2.plane.invert();

    p1.isInvert = true;
    p2.isInvert = false;

    if (crd1.isRight()) p2.polygon = std::vector<glm::vec4>(p2.polygon.rbegin(),
    ↪  p2.polygon.rend());
    if (crd2.isRight()) p1.polygon = std::vector<glm::vec4>(p1.polygon.rbegin(),
    ↪  p1.polygon.rend());

    p1.isTeleportInvert = crd1.isRight() ^ crd2.isRight();
    p2.isTeleportInvert = crd1.isRight() ^ crd2.isRight();

    p1.color = spob2glm(clr1);
    p2.color = spob2glm(clr2);

    return {p1, p2};
}

//------------------------------------------------------------------------------
//------------------------------------------------------------------------------
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
glm::mat4 getFromMatrix(const spob::crd3& crd) {
    glm::mat4 result;
    result[0] = glm::vec4(crd.i.x, crd.j.x, crd.k.x, -crd.pos.x);
    result[1] = glm::vec4(crd.i.y, crd.j.y, crd.k.y, -crd.pos.y);
    result[2] = glm::vec4(crd.i.z, crd.j.z, crd.k.z, -crd.pos.z);
    result[3] = glm::vec4(0, 0, 0, -1);
    return glm::transpose(result);
}

//------------------------------------------------------------------------------
glm::mat4 getToMatrix(const spob::crd3& crd) {
    return glm::inverse(getFromMatrix(crd));
}

//------------------------------------------------------------------------------
glm::vec4 spob2glm(const spob::vec2& vec) {
    return {vec.x, vec.y, 0, 1};
}

//------------------------------------------------------------------------------
glm::vec4 spob2glm(const spob::vec3& vec) {
    return {vec.x, vec.y, vec.z, 1};
}

//------------------------------------------------------------------------------
std::vector<glm::vec4> spob2glm(const std::vector<spob::vec3>& mas) {
    std::vector<glm::vec4> result;
    for (const auto& i : mas)
```

```
356         result.push_back(spob2glm(i));
357     return result;
358 }
359 
360 //-----------------------------------------------------------------------------
361 std::vector<glm::vec4> spob2glm(const std::vector<spob::vec2>& mas) {
362     std::vector<glm::vec4> result;
363     for (const auto& i : mas)
364         result.push_back(spob2glm(i));
365     return result;
366 }
367 
368 //-----------------------------------------------------------------------------
369 std::vector<glm::vec4> spob2glm(const std::vector<spob::vec2>& mas, const
    ↪  spob::plane3& plane) {
370     std::vector<glm::vec4> result;
371     for (const auto& i : mas)
372         result.push_back(spob2glm(plane.from(i)));
373     return result;
374 }
375 
376 //-----------------------------------------------------------------------------
377 glm::vec3 spheric2cartesian(glm::vec3 spheric) {
378     auto& alpha = spheric.x;
379     auto& beta = spheric.y;
380     auto& r = spheric.z;
381     return glm::vec3(
382         r * sin(beta) * cos(alpha),
383         r * sin(beta) * sin(alpha),
384         r * cos(beta)
385     );
386 }
387 
388 //-----------------------------------------------------------------------------
389 glm::vec3 cartesian2spheric(glm::vec3 cartesian) {
390     auto& x = cartesian.x;
391     auto& y = cartesian.y;
392     auto& z = cartesian.z;
393     return glm::vec3(
394         std::atan2(y, x),
395         std::atan2(std::sqrt(x*x + y*y), z),
396         std::sqrt(x*x + y*y + z*z)
397     );
398 }
399 
400 //-----------------------------------------------------------------------------
401 std::vector<glm::vec4> projectPolygonToScreen(const std::vector<glm::vec4>&
    ↪  polygon) {
402     std::array<GLdouble, 16> projection;
403     std::array<GLdouble, 16> modelview;
404     std::array<GLdouble, 3>  projected;
405     std::array<GLint, 4>     viewport;
406 
407     glGetDoublev(GL_PROJECTION_MATRIX, projection.data());
408     glGetDoublev(GL_MODELVIEW_MATRIX, modelview.data());
409     glGetIntegerv(GL_VIEWPORT, viewport.data());
410 
411     std::vector<glm::vec4> result;
412     for (auto& i : polygon) {
```

```
        gluProject(i.x, i.y, i.z,
                   modelview.data(), projection.data(), viewport.data(),
                   &projected[0], &projected[1], &projected[2]);
        result.push_back(glm::vec4(projected[0], projected[1], projected[2],
        ↪  1.0));
    }

    return result;
}

//----------------------------------------------------------------------------
std::vector<std::vector<glm::vec4>> intersect(const
↪  std::vector<std::vector<glm::vec4>>& a, const std::vector<glm::vec4>& b) {
    std::vector<std::vector<glm::vec4>> result;

    if (a.empty() || b.empty())
        return result;

    using namespace ClipperLib;
    Path clip;
    Paths subj(a.size());
    Paths solution;

    for (int i = 0; i < a.size(); ++i) {
        for (const auto& j : a[i]) {
            subj[i].push_back(IntPoint(j.x, j.y));
        }
    }
    for (const auto& i : b) {
        clip.push_back(IntPoint(i.x, i.y));
    }

    Clipper c;
    c.AddPaths(subj, ptSubject, true);
    c.AddPath(clip, ptClip, true);
    c.Execute(ctIntersection, solution, pftNonZero, pftNonZero);

    for (auto& i : solution) {
        result.push_back({});
        for (auto& j : i)
            result.back().push_back(glm::vec4(j.X, j.Y, 0, 1));
    }

    return result;
}
```

FILE **plane.cpp**

```
#include <GL/glew.h>
#include <glm/gtc/type_ptr.hpp>
#include <prtl_vis/plane.h>

std::vector<Plane> ClipPlane::p_stack;

//----------------------------------------------------------------------------
void Plane::invert(void) {
    x = -x;
    y = -y;
    z = -z;
    w = -w;
```

```
}

//----------------------------------------------------------------------------
void ClipPlane::activate(const Plane& p) {
    if (!p_stack.empty())
        glDisable(GL_CLIP_PLANE0);

    p_stack.push_back(p);
    GLdouble plane[4] = {p.x, p.y, p.z, p.w};
    glClipPlane(GL_CLIP_PLANE0, plane);
    glEnable(GL_CLIP_PLANE0);
}

//----------------------------------------------------------------------------
void ClipPlane::disable(void) {
    glDisable(GL_CLIP_PLANE0);
    p_stack.pop_back();

    if (!p_stack.empty()) {
        Plane p = p_stack.back();
        GLdouble plane[4] = {p.x, p.y, p.z, p.w};
        glClipPlane(GL_CLIP_PLANE0, plane);
        glEnable(GL_CLIP_PLANE0);
    }
}

//----------------------------------------------------------------------------
Plane ClipPlane::getCurrentPlane(void) {
    if (p_stack.empty())
        throw std::exception();
    return p_stack.back();
}

//----------------------------------------------------------------------------
bool isPointBehindPlane(const Plane& plane, const glm::vec4& point) {
    double planeValue =
        plane.x*point.x +
        plane.y*point.y +
        plane.z*point.z +
        plane.w;
    return planeValue > 0.001;
}

//----------------------------------------------------------------------------
bool isPolygonBehindPlane(const Plane& plane, const std::vector<glm::vec4>&
↪   polygon) {
    bool isBehindPlane = true;
    for (auto& i : polygon)
        isBehindPlane &= !isPointBehindPlane(plane, i);
    return !isBehindPlane;
}

//----------------------------------------------------------------------------
glm::vec4 getClipPlaneEquation(void) {
    auto plane = ClipPlane::getCurrentPlane();
    glm::dmat4 modelview;
    glGetDoublev(GL_MODELVIEW_MATRIX, glm::value_ptr(modelview));

    auto result = plane * glm::inverse(modelview);
```

```
    return result;
}
```

FILE **scene_reader.cpp**

```cpp
#include <prtl_vis/scene_reader.h>

namespace scene
{

//-----------------------------------------------------------------------------
Scene parseScene(const json& obj) {
    Scene result;
    result.cam_rotate_around = parseVec3(obj["cam_rotate_around"]);
    result.cam_spheric_pos = parseVec3(obj["cam_spheric_pos"]);
    if (obj.find("frames") != obj.end())
        for (const auto& i : obj["frames"])
            result.frames.push_back(parseFrame(i));
    return result;
}

//-----------------------------------------------------------------------------
Frame parseFrame(const json& obj) {
    Frame result;
    if (obj.find("colored_polygons") != obj.end())
        for (const auto& i : obj["colored_polygons"])
            result.colored_polygons.push_back(parseColoredPolygon(i));
    if (obj.find("textured_polygons") != obj.end())
        for (const auto& i : obj["textured_polygons"])
            result.textured_polygons.push_back(parseTexturedPolygon(i));
    if (obj.find("portals") != obj.end())
        for (const auto& i : obj["portals"])
            result.portals.push_back(parsePortal(i));
    if (obj.find("textures") != obj.end())
        for (const auto& i : obj["textures"])
            result.textures.push_back(parseTexture(i));
    if (obj.find("luminaries") != obj.end())
        for (const auto& i : obj["luminaries"])
            result.luminaries.push_back(parseLuminary(i));
    return result;
}

//-----------------------------------------------------------------------------
Luminary parseLuminary(const json& obj) {
    Luminary result;
    result.pos = parseVec3(obj["pos"]);
    result.color = parseVec3(obj["color"]);
    return result;
}

//-----------------------------------------------------------------------------
Texture parseTexture(const json& obj) {
    Texture result;
    result.filename = obj["filename"].get<std::string>();
    result.id = obj["id"];
    return result;
}

//-----------------------------------------------------------------------------
```

```
TexturedPolygon parseTexturedPolygon(const json& obj) {
    TexturedPolygon result;
    result.crd = parseCrd3(obj["crd"]);
    result.texture_id = obj["texture_id"];
    if (obj.find("polygon") != obj.end())
        for (const auto& i : obj["polygon"])
            result.polygon.push_back(parseVec2(i));
    if (obj.find("tex_coords") != obj.end())
        for (const auto& i : obj["tex_coords"])
            result.tex_coords.push_back(parseVec2(i));
    return result;
}

//-----------------------------------------------------------------------------
ColoredPolygon parseColoredPolygon(const json& obj) {
    ColoredPolygon result;
    result.crd = parseCrd3(obj["crd"]);
    result.color = parseVec3(obj["color"]);
    if (obj.find("polygon") != obj.end())
        for (const auto& i : obj["polygon"])
            result.polygon.push_back(parseVec2(i));
    return result;
}

//-----------------------------------------------------------------------------
Portal parsePortal(const json& obj) {
    Portal result;
    result.crd1 = parseCrd3(obj["crd1"]);
    result.crd2 = parseCrd3(obj["crd2"]);
    result.color1 = parseVec3(obj["color1"]);
    result.color2 = parseVec3(obj["color2"]);
    if (obj.find("polygon") != obj.end())
        for (const auto& i : obj["polygon"])
            result.polygon.push_back(parseVec2(i));
    return result;
}

//-----------------------------------------------------------------------------
spob::crd3 parseCrd3(const json& obj) {
    spob::crd3 result;
    result.i = parseVec3(obj["i"]);
    result.j = parseVec3(obj["j"]);
    result.k = parseVec3(obj["k"]);
    result.pos = parseVec3(obj["pos"]);
    return result;
}

//-----------------------------------------------------------------------------
spob::vec3 parseVec3(const json& obj) {
    spob::vec3 result;
    result.x = obj[0];
    result.y = obj[1];
    result.z = obj[2];
    return result;
}

//-----------------------------------------------------------------------------
spob::vec2 parseVec2(const json& obj) {
    spob::vec2 result;
```

```
    result.x = obj[0];
    result.y = obj[1];
    return result;
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
json unparse(const Scene& scene) {
    json result;
    result["cam_rotate_around"] = unparse(scene.cam_rotate_around);
    result["cam_spheric_pos"] = unparse(scene.cam_spheric_pos);
    for (auto& i : scene.frames)
        result["frames"].push_back(unparse(i));
    return result;
}

//-----------------------------------------------------------------------------
json unparse(const Frame& frame) {
    json result;
    result["textured_polygons"] = {};
    result["colored_polygons"] = {};
    result["portals"] = {};
    for (auto& i : frame.textured_polygons)
        result["textured_polygons"].push_back(unparse(i));
    for (auto& i : frame.colored_polygons)
        result["colored_polygons"].push_back(unparse(i));
    for (auto& i : frame.portals)
        result["portals"].push_back(unparse(i));
    for (auto& i : frame.textures)
        result["textures"].push_back(unparse(i));
    for (auto& i : frame.luminaries)
        result["luminaries"].push_back(unparse(i));
    return result;
}

//-----------------------------------------------------------------------------
json unparse(const Luminary& luminary) {
    json result;
    result["pos"] = unparse(luminary.pos);
    result["color"] = unparse(luminary.color);
    return result;
}

//-----------------------------------------------------------------------------
json unparse(const Texture& texture) {
    json result;
    result["filename"] = texture.filename;
    result["id"] = texture.id;
    return result;
}

//-----------------------------------------------------------------------------
json unparse(const TexturedPolygon& textured_polygon) {
    json result;
    result["crd"] = unparse(textured_polygon.crd);
    result["texture_id"] = textured_polygon.texture_id;
```

```
    for (auto& i : textured_polygon.polygon)
        result["polygon"].push_back(unparse(i));
    for (auto& i : textured_polygon.tex_coords)
        result["tex_coords"].push_back(unparse(i));
    return result;
}

//------------------------------------------------------------------------------
json unparse(const ColoredPolygon& colored_polygon) {
    json result;
    result["crd"] = unparse(colored_polygon.crd);
    result["color"] = unparse(colored_polygon.color);
    for (auto& i : colored_polygon.polygon)
        result["polygon"].push_back(unparse(i));
    return result;
}

//------------------------------------------------------------------------------
json unparse(const Portal& portal) {
    json result;
    result["crd1"] = unparse(portal.crd1);
    result["crd2"] = unparse(portal.crd2);
    result["color1"] = unparse(portal.color1);
    result["color2"] = unparse(portal.color2);
    for (auto& i : portal.polygon)
        result["polygon"].push_back(unparse(i));
    return result;
}

//------------------------------------------------------------------------------
json unparse(const spob::crd3& crd) {
    json result;
    result["i"] = unparse(crd.i);
    result["j"] = unparse(crd.j);
    result["k"] = unparse(crd.k);
    result["pos"] = unparse(crd.pos);
    return result;
}

//------------------------------------------------------------------------------
json unparse(const spob::vec3& vec) {
    json result;
    result.push_back(vec.x);
    result.push_back(vec.y);
    result.push_back(vec.z);
    return result;
}

//------------------------------------------------------------------------------
json unparse(const spob::vec2& vec) {
    json result;
    result.push_back(vec.x);
    result.push_back(vec.y);
    return result;
}

};
```

FILE **shader.cpp**

```cpp
#include <stdio.h>
#include <string>
#include <vector>
#include <iostream>
#include <fstream>
#include <algorithm>
#include <sstream>

using namespace std;

#include <stdlib.h>
#include <string.h>

#include <GL/glew.h>
#include <prtl_vis/shader.h>

//----------------------------------------------------------------------------
GLuint LoadShaders(const char * vertex_file_path,const char *
 ↪  fragment_file_path){

    // Create the shaders
    GLuint VertexShaderID = glCreateShader(GL_VERTEX_SHADER);
    GLuint FragmentShaderID = glCreateShader(GL_FRAGMENT_SHADER);

    // Read the Vertex Shader code from the file
    std::string VertexShaderCode;
    std::ifstream VertexShaderStream(vertex_file_path, std::ios::in);
    if(VertexShaderStream.is_open()){
        std::stringstream sstr;
        sstr << VertexShaderStream.rdbuf();
        VertexShaderCode = sstr.str();
        VertexShaderStream.close();
    }else{
        printf("Impossible to open %s. Are you in the right directory ? Don't
         ↪  forget to read the FAQ !\n", vertex_file_path);
        getchar();
        return 0;
    }

    // Read the Fragment Shader code from the file
    std::string FragmentShaderCode;
    std::ifstream FragmentShaderStream(fragment_file_path, std::ios::in);
    if(FragmentShaderStream.is_open()){
        std::stringstream sstr;
        sstr << FragmentShaderStream.rdbuf();
        FragmentShaderCode = sstr.str();
        FragmentShaderStream.close();
    }

    GLint Result = GL_FALSE;
    int InfoLogLength;


    // Compile Vertex Shader
    printf("Compiling shader : %s\n", vertex_file_path);
    char const * VertexSourcePointer = VertexShaderCode.c_str();
    glShaderSource(VertexShaderID, 1, &VertexSourcePointer , NULL);
    glCompileShader(VertexShaderID);
```

```
57
58      // Check Vertex Shader
59      glGetShaderiv(VertexShaderID, GL_COMPILE_STATUS, &Result);
60      glGetShaderiv(VertexShaderID, GL_INFO_LOG_LENGTH, &InfoLogLength);
61      if ( InfoLogLength > 0 ){
62          std::vector<char> VertexShaderErrorMessage(InfoLogLength+1);
63          glGetShaderInfoLog(VertexShaderID, InfoLogLength, NULL,
            ↪  &VertexShaderErrorMessage[0]);
64          printf("%s\n", &VertexShaderErrorMessage[0]);
65      }
66
67
68
69      // Compile Fragment Shader
70      printf("Compiling shader : %s\n", fragment_file_path);
71      char const * FragmentSourcePointer = FragmentShaderCode.c_str();
72      glShaderSource(FragmentShaderID, 1, &FragmentSourcePointer , NULL);
73      glCompileShader(FragmentShaderID);
74
75      // Check Fragment Shader
76      glGetShaderiv(FragmentShaderID, GL_COMPILE_STATUS, &Result);
77      glGetShaderiv(FragmentShaderID, GL_INFO_LOG_LENGTH, &InfoLogLength);
78      if ( InfoLogLength > 0 ){
79          std::vector<char> FragmentShaderErrorMessage(InfoLogLength+1);
80          glGetShaderInfoLog(FragmentShaderID, InfoLogLength, NULL,
            ↪  &FragmentShaderErrorMessage[0]);
81          printf("%s\n", &FragmentShaderErrorMessage[0]);
82      }
83
84
85
86      // Link the program
87      printf("Linking program\n");
88      GLuint ProgramID = glCreateProgram();
89      glAttachShader(ProgramID, VertexShaderID);
90      glAttachShader(ProgramID, FragmentShaderID);
91      glLinkProgram(ProgramID);
92
93      // Check the program
94      glGetProgramiv(ProgramID, GL_LINK_STATUS, &Result);
95      glGetProgramiv(ProgramID, GL_INFO_LOG_LENGTH, &InfoLogLength);
96      if ( InfoLogLength > 0 ){
97          std::vector<char> ProgramErrorMessage(InfoLogLength+1);
98          glGetProgramInfoLog(ProgramID, InfoLogLength, NULL,
            ↪  &ProgramErrorMessage[0]);
99          printf("%s\n", &ProgramErrorMessage[0]);
100     }
101
102
103     glDetachShader(ProgramID, VertexShaderID);
104     glDetachShader(ProgramID, FragmentShaderID);
105
106     glDeleteShader(VertexShaderID);
107     glDeleteShader(FragmentShaderID);
108
109     return ProgramID;
110 }
```

5.3 Шейдеры

FILE **draw.fragment.glsl**

```
#version 330 core

out vec4 color;

uniform sampler2D Color;
uniform sampler2D Depth;

void main(){
    ivec2 texcoord = ivec2(floor(gl_FragCoord.xy));

    color = texelFetch(Color, texcoord, 0);
    gl_FragDepth = texelFetch(Depth, texcoord, 0).r;
}
```

FILE **draw.vertex.glsl**

```
#version 110

void main()
{
    gl_Position = gl_Vertex;
}
```

FILE **drawpoly.fragment.glsl**

```
#version 330 core

out vec4 color;

uniform sampler2D Color;
uniform sampler2D Depth;

void main(){
    ivec2 texcoord = ivec2(floor(gl_FragCoord.xy));

    color = texelFetch(Color, texcoord, 0);
    gl_FragDepth = gl_FragCoord.z;
}
```

FILE **drawpoly.vertex.glsl**

```
#version 110

void main()
{
    gl_Position = ftransform();

    // fix of the clipping bug for both Nvidia and ATi
    #ifdef __GLSL_CG_DATA_TYPES
    gl_ClipVertex = gl_ModelViewMatrix * gl_Vertex;
    #endif
}

// Source: https://forums.khronos.org/showthread.php/68274-How-to-activate-clip-planes-via-shader?p=331885&viewfull=1#post331885 . Thank you, ehsan2004!
```

FILE **merge.fragment.glsl**

```
#version 330 core

out vec4 color;

uniform sampler2D Color1;
uniform sampler2D Depth1;
uniform sampler2D Color2;
uniform sampler2D Depth2;

void main(){
    ivec2 texcoord = ivec2(floor(gl_FragCoord.xy));

    vec4 color1 = texelFetch(Color1, texcoord, 0);
    float depth1 = texelFetch(Depth1, texcoord, 0).r;
    vec4 color2 = texelFetch(Color2, texcoord, 0);
    float depth2 = texelFetch(Depth2, texcoord, 0).r;

    if (depth1 < depth2) {
        //color = (color1 + vec4(vec3(depth1), 1.0))/2.0;
        color = color1;
        gl_FragDepth = depth1;
    } else {
        //color = (color2 + vec4(vec3(depth2), 1.0))/2.0;
        color = color2;
        gl_FragDepth = depth2;
    }
}
```

FILE **merge.vertex.glsl**

```
#version 110

void main()
{
    gl_Position = gl_Vertex;
}
```

5.4 Сцены

В целях экономия места написана только одна сцена:

FILE **volumetric_portal_with_textures.json**

```
{
    ”cam_rotate_around”:[0.5,0.9666666666666666,0.5],
    ”cam_spheric_pos”:[0.54,1.2,1.7],
    ”frames”:[{
        ”luminaries”:[{
            ”pos”:[3,3,1],
            ”color”:[1,1,1]
        },{
            ”pos”:[-4,-4,4],
            ”color”:[0.5,0.5,0.5]
        }],
        ”colored_polygons”:[
```

```
            {"color":[0.1,0.55,1.0],"crd":{"i":[0.8660254037844387,0.49999999999999994,0.0],"j":[0.0,0.0,1.0],"k":[-0.49999999999999994,0.8660254037844387,0.0],"pos":[0.0,0.0,0.0]},"polygon":[[0.5773502691896257,0.0],[0.5773502691896257,-0.03],[-0.030000000000000027,-0.03],[-0.03,1.0299999999999998],[0.5773502691896257,1.03],[0.5773502691896257,1.0],[0.0,1.0],[0.0,0.0]]},
            {"color":[1.0,0.5,0.15],"crd":{"i":[0.8660254037844387,0.49999999999999994,0.0],"j":[0.0,0.0,1.0],"k":[-0.49999999999999994,0.8660254037844387,0.0],"pos":[0.0,1.5,0.0]},"polygon":[[0.5773502691896257,0.0],[0.5773502691896257,-0.03],[-0.030000000000000027,-0.03],[-0.03,1.0299999999999998],[0.5773502691896257,1.03],[0.5773502691896257,1.0],[0.0,1.0],[0.0,0.0]]},
            {"color":[0.1,0.55,1.0],"crd":{"i":[0.8660254037844387,-0.5,0.0],"j":[0.0,0.0,1.0],"k":[0.5,0.8660254037844387,0.0],"pos":[0.5,0.2886751345948128,0.0]},"polygon":[[0.0,1.0],[0.0,1.03],[0.6073502691896258,1.03],[0.6073502691896258,-0.029999999999999916],[0.0,-0.03],[0.0,0.0],[0.5773502691896257,0.0],[0.5773502691896257,1.0]]},
            {"color":[1.0,0.5,0.15],"crd":{"i":[0.8660254037844387,-0.5,0.0],"j":[0.0,0.0,1.0],"k":[0.5,0.8660254037844387,0.0],"pos":[0.5,1.7886751345948129,0.0]},"polygon":[[0.0,1.0],[0.0,1.03],[0.6073502691896258,1.03],[0.6073502691896258,-0.029999999999999916],[0.0,-0.03],[0.0,0.0],[0.5773502691896257,0.0],[0.5773502691896257,1.0]]}],
        "portals":[{"color1":[1.0,0.5,0.15],"color2":[0.1,0.55,1.0],"crd1":{"i":[0.8660254037844387,0.49999999999999994,0.0],"j":[0.0,0.0,1.0],"k":[-0.49999999999999994,0.8660254037844387,0.0],"pos":[0.0,0.0,0.0]},"crd2":{"i":[0.8660254037844387,0.49999999999999994,0.0],"j":[0.0,0.0,1.0],"k":[-0.49999999999999994,0.8660254037844387,0.0],"pos":[0.0,1.5,0.0]},"polygon":[[0.5773502691896257,0.0],[0.0,0.0],[0.0,1.0],[0.5773502691896257,1.0]]},{"color1":[1.0,0.5,0.15],"color2":[0.1,0.55,1.0],"crd1":{"i":[0.8660254037844387,-0.5,0.0],"j":[0.0,0.0,1.0],"k":[0.5,0.8660254037844387,0.0],"pos":[0.5,0.2886751345948128,0.0]},"crd2":{"i":[0.8660254037844387,-0.5,0.0],"j":[0.0,0.0,1.0],"k":[0.5,0.8660254037844387,0.0],"pos":[0.5,1.7886751345948129,0.0]},"polygon":[[0.0,1.0],[0.5773502691896257,1.0],[0.5773502691896257,0.0],[0.0,0.0]]}],
        "textured_polygons":[
            {"texture_id":6,"tex_coords":[[0.5,0.0234375],[0.0234375,0.96875],[0.96875,0.96875]],"crd":{"i":[1.0,0.0,0.0],"j":[0.0,1.0,0.0],"k":[0.0,0.0,1.0],"pos":[0.5,0.5,0.5]},"polygon":[[0.0,0.0],[-0.35,0.7],[0.35,0.7]]},
            {"texture_id":0,"tex_coords":[[0,0],[5,0],[5,5],[0,5]],"crd":{"i":[5.0,0.0,0.0],"j":[0.0,5.0,0.0],"k":[0.0,0.0,5.0],"pos":[0.0,0.0,5.0]},"polygon":[[-1.0,-1.0],[-1.0,1.0],[1.0,1.0],[1.0,-1.0]]},
            {"texture_id":1,"tex_coords":[[0,0],[5,0],[5,5],[0,5]],
            "crd":{"i":[5.0,0.0,0.0],"j":[0.0,5.0,0.0],"k":[0.0,0.0,5.0],"pos":[-0.0,-0.0,-5.0]},"polygon":[[-1.0,-1.0],[-1.0,1.0],[1.0,1.0],[1.0,-1.0]]},
            {"texture_id":2,"tex_coords":[[0,0],[5,0],[5,5],[0,5]], "crd":{"i":[5.0,0.0,0.0],"j":[0.0,0.0,5.0],"k":[0.0,5.0,0.0],"pos":[0.0,5.0,0.0]},"polygon":[[-1.0,-1.0],[-1.0,1.0],[1.0,1.0],[1.0,-1.0]]},
            {"texture_id":3,"tex_coords":[[0,0],[5,0],[5,5],[0,5]],
            "crd":{"i":[5.0,0.0,0.0],"j":[0.0,0.0,5.0],"k":[0.0,5.0,0.0],"pos":[-0.0,-5.0,-0.0]},"polygon":[[-1.0,-1.0],[-1.0,1.0],[1.0,1.0],[1.0,-1.0]]},
            {"texture_id":4,"tex_coords":[[0,0],[5,0],[5,5],[0,5]], "crd":{"i":[0.0,0.0,5.0],"j":[0.0,5.0,0.0],"k":[5.0,0.0,0.0],"pos":[5.0,0.0,0.0]},"polygon":[[-1.0,-1.0],[-1.0,1.0],[1.0,1.0],[1.0,-1.0]]},
```

```
            {"texture_id":5,"tex_coords":[[0,0],[5,0],[5,5],[0,5]],
            ↪  "crd":{"i":[0.0,0.0,5.0],"j":[0.0,5.0,0.0],"k":[5.0,0.0,0.0],"po
            ↪  s":[-5.0,-0.0,-0.0]},"polygon":[[-1.0,-1.0],[-1.0,1.0],[1.0,1.0]
            ↪  ,[1.0,-1.0]]}],
        "textures":[
            {"filename":"img/1.png", "id":0},
            {"filename":"img/2.png", "id":1},
            {"filename":"img/3.png", "id":2},
            {"filename":"img/4.png", "id":3},
            {"filename":"img/5.png", "id":4},
            {"filename":"img/6.png", "id":5},
            {"filename":"img/7.png", "id":6}]
    }]
}
```