СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Воздушный транспорт является одним из самых безопасных и востребованных видов транспорта. На сегодняшний день это не только способ доставки грузов и пассажиров в отдаленные и труднодоступные участки по всей планете, но и стратегически и экономически важная отрасль государства. Непрерывное развитие и охват новых территорий и направлений, влечет за собой стремительный рост количества воздушных судов, одновременно находящихся в небе. С повышением интенсивности воздушного движения растет потребность в его упорядочивании.

Создание условий, при которых для каждого эксплуатанта воздушного пространства минимизируется риск столкновения и возможные эксплуатационные потери, основная задача организации воздушного движения. Для решения этой задачи, а также снижения экономических и временных затрат авиаперевозчиков и пассажиров необходимо использование грамотно спроектированной структуры воздушного пространства, основными элементами которой являются:

а) Контролируемое воздушное пространство (районы аэродромов, секторы обслуживания воздушного движения);

б) Маршруты обслуживания воздушного движения (воздушные трассы и маршруты вылета и прибытия воздушных судов);

в) Зоны ограничения полетов и запретные зоны;

г) Естественные и искусственные препятствия (рельеф, растительность, строения).

В значительной степени, сложность изучения и анализа существующей структуры воздушного транспорта, по сравнению с инфраструктурой наземного транспорта, заключается в том, что она является невидимой для человека. В настоящий момент данные обо всех элементах структуры воздушного пространства представлены во множестве разрозненных источников, таких как многотомные аэронавигационные сборники, карты, аэронавигационные базы данных. Из-за специфики существующего формата публикации документов аэронавигационной информации, данные об интересующих элементах структуры воздушного пространства могут быть разнесены на разные страницы сборника, а карты предоставляют ограниченную вариативность отображения сочетаемых объектов. Более того, даже использование электронных аэронавигационных баз данных не позволяет в полной мере получить полное представление о взаимном расположении элементов воздушного пространства, например, для нескольких близкорасположенных аэродромов, поскольку инструмента, предоставляющего возможность одновременно отобразить в трехмерном пространстве такие элементы как процедуры вылета и прибытия, в настоящий момент просто не существует.

Данный подход хранения данных существенно повышает сложность восприятия взаимного расположения объектов структуры в целом и понимание взаиморасположения разнотипных объектов ВП вблизи аэронавигационных точек.

Несмотря на тенденцию к автоматизации системы управления воздушным движением, полностью исключить человеческий фактор в системе на данный момент не представляется возможным. Постоянный контроль корректности движения самолетов должен осуществляться авиадиспетчером. В своей зоне ответственности он обеспечивает безопасное и эффективное использование воздушного пространства для всех его участников. Таким образом, для выполнения своих обязанностей, авиадиспетчер должен знать и хорошо

представлять структуру воздушного пространства, в которой он обслуживает движение воздушных судов.

При внедрении новой версии или изучении существующей структуры воздушного пространства, перед авиадиспетчером стоит задача анализа взаиморасположения различных сочетаний элементов в его зоне ответственности. Учитывая способ хранения информации об элементах, поиск данных и представление структуры воздушного пространства требуют больших временных и трудовых затрат.

На текущий момент не существует никаких средств полноценной визуализации всех элементов структуры воздушного пространства. На международном рынке представлены лишь частные решения для визуализации отдельных типов элементов. Тем не менее, уже разработана модель обмена аэронавигационной информацией (AIXM 5.1.1), которая активно используется проектировщиками структур воздушного пространства.

Для проверки разработанной структуры на работоспособность, подготовки диспетчерского персонала к работе в новых условиях, а также моделирования различных вероятных ситуаций при обеспечении навигации в воздушном пространстве применяются диспетчерские тренажеры. Тренажеры позволяют повысить эффективность подготовки диспетчеров, при этом моделирование нештатных ситуаций не влечет за собой возникновения каких-либо рисков. Также это позволяет свести к минимуму материальные затраты, создавая эффективную среду для получения практического опыта, максимально приближенного к реальному.

Фирма “НИТА” на сегодняшний день является одним из самых известных среди российских и зарубежных предприятий, ведущих разработку и производство оборудования и программного обеспечения для осуществления проектирования и обслуживания движения в воздушном пространстве.

На данный момент в тренажерах и применяемом оборудовании диспетчерских пунктов фирмы отображение структуры и текущей воздушной

обстановки в зоне ответственности авиадиспетчера производится в двухмерном формате. Данный подход упрощает визуализацию, однако требует от пользователя определенных навыков работы с системой, пространственного воображения и предварительного тщательного изучения структуры ВП.

Значительно снизить сложность восприятия, а также трудовые и временные затраты при обучении стажеров могла бы возможность использования в качестве справочной информации трехмерной визуализации объема пространства в предполагаемой или моделируемой зоне ответственности авиадиспетчера.

Учитывая все выявленные сложности, очевидным решением стала разработка и дальнейшее внедрение в тренажер нового графического движка, способного производить визуализацию структуры воздушного пространства. Характерной особенностью данного компонента должна быть возможность представления моделей объектов структуры с высокой точностью привязки по географическим координатам.

В основе визуализации двухмерной и трехмерной компьютерной графики лежит графический движок - промежуточное программное обеспечение или рендерер. В зависимости от задачи отрисовки, движки могут быть как реального времени, так и с долговременной обработкой кадра. Несмотря на разнообразие доступных в настоящий момент свободно распространяемых и платных движков, объем отображаемого пространства, построенного на точных географических данных таких решений, строго ограничен несколькими десятками квадратных километров. При этом основной акцент при разработке существующих рендереров ставится на увеличение реалистичности визуальных эффектов и качестве отображения трехмерных моделей объектов.

Целью данной работы является проектирование и разработка трехмерного графического движка для визуализатора воздушного пространства, встраиваемого в качестве справочной информации в диспетчерский тренажер «ЭКСПЕРТ».

ЗАКЛЮЧЕНИЕ

Основная задача выпускной квалификационной работы состояла в проектировании и разработке графического движка для трехмерного визуализатора структуры воздушного пространства.

В рамках работы был произведен обзор предметной области, описана системная архитектура визуализатора и на ее основе выдвинуты требования к графическому движку. Проведен анализ возможных вариантов решения задачи.

В результате работы был разработан и реализован графический движок для трехмерного визуализатора воздушного пространства.

Разработанный функционал графического движка визуализатора продолжает наращиваться, однако уже сейчас имеется возможность отображения всех требуемых элементов структуры воздушного пространства, что позволяет использовать визуализатор отделу ОрВД для анализа и тестирования разрабатываемых структур воздушного пространства.

Дальнейшее развитие данного движка заключается в его подготовке к встраиванию в диспетчерские тренажеры фирмы “НИТА” в качестве справочной и обучающей системы.

Таким образом, задачу выпускной квалификационной работы можно считать выполненной.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Буч Г., Якобсон И., Рамбо Д. Язык UML. Руководство пользователя. // СПб.: Питер. – 2014. – 496 с.
2. Герберт Шилдт C++ базовый курс https://www.bsuir.by/m/12_100229_1_98220.pdf. - 2008.
3. Дж. Рамбо, М. Блаха UML 2.0. Объектно-ориентированное моделирование и разработка // Питер 2007. – 544 с.
4. Шлее М Qt 5.3. Профессиональное программирование на C++ // СПб.: БХВ-Петербург. – 2015. - 929
5. AIXM (Aeronautical Information Exchange Model) http://www.aixm.aero/sites/aixm.aero/files/imce/AIXM511HTML/index.html.
6. Bjarne Stroustrup The C++ Programming Language, 4th Edition // Addison-Wesley Professional. – 2013. – 1368 с.
7. Dave Shreiner OpenGL Programming Guide Seventh Edition // Addison-wesley. - 2010. - 885 с.
8. Guillaume Lazar Mastering Qt 5 // Packt Publishing. – 2016. – 575 с.
9. J. Gregor Game Engine Architecture 2st Edition. // CRCPres. - 2014. - 1052 с.
10. Joey de Vries Learn OpenGL. https://learnopengl.com/book/offline%20learnopengl.pdf. - 2015.
11. Olsen J. Realtime procedural terrain generation. http://web.mit.edu/cesium/Public/terrain.pdf .pdf. – 2004
12. M. McShaffry Game Coding Complete, 3rd Edition. - 2003.- 944 с.
13. Nick Brettell Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids http://hhoppe.com/geomclipmap.pdf. - 2005.
14. Peter Shirley, Fundamentals of Computer Graphics 3rd Edition // A K Peters/CRC Press. – 2009. – 804 с.
15. Robert Nystrom Game Programming Patterns. - 2014. – 354 с.
16. W. Wysota Game Programming Using QT // Packt Pub. – 2016. – 512 с.

ПРИЛОЖЕНИЕ

Приложение А

```
#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec4 color;
layout (location = 2) in vec2 texCoords;
layout (location = 3) in vec3 normal;
layout (location = 5) in vec3 heightVec;

out vec3 Normal;
out vec3 FragPos;
out vec2 TexCoords;
out vec4 ourColor;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

uniform sampler2D heightmap;
uniform bool useHeightMap;
uniform bool useHeightVec;
uniform float heightRate;

uniform int minHeight = 0;
uniform int maxHeight = 8848;

void main()
{
    float posMinHeight = minHeight / 8848.0 ;
    float posMaxHeight = maxHeight / 8848.0 ;

    float h = (posMaxHeight - posMinHeight)*texture2D(heightmap, texCoords).r + posMinHeight;

    vec3 position1 = vec3(position.x * h, position.y * h, position.z * h);
    vec3 normalPosition = normalize(position);
    h = h * heightRate;
    vec3 addposition = vec3(normalPosition.x * h,normalPosition.y * h, normalPosition.z * h);

    if (useHeightVec) {
        vec3 addposition1 = vec3(heightVec.x * heightRate,
                                 heightVec.y * heightRate,
                                 heightVec.z * heightRate);

        gl_Position = projection * view *  model * vec4(position + addposition1, 1.0f);
    } else  if (useHeightMap){
        gl_Position = projection * view *  model * vec4(position + addposition, 1.0f);
    } else {
        gl_Position = projection * view *  model * vec4(position, 1.0f);
    }

    FragPos = vec3(model * vec4(position, 1.0f));
    Normal = mat3(transpose(inverse(model))) * normal;

    ourColor = color;
    TexCoords = texCoords;
}
```

```
layout (location = 0) out vec4 FragColor;
layout (location = 1) out vec4 BrightColor;

in vec3 Normal, FragPos;
in vec2 TexCoords;
in vec4 ourColor;

uniform vec3 viewPos;
uniform vec3 lightColor;
uniform vec3 objectColor;
uniform Light light;
uniform Material material;
uniform bool isBloom;

struct Light {
    vec3 position;
    vec3 ambient;
    vec3 diffuse;
    vec3 specular;

    float constant;
    float linear;
    float quadratic;
};

struct Material {
    sampler2D diffuse;

    vec3      ambient;
    vec3      specular;
    float     shininess;
};

void main()
{
    vec3 ambient, diffuse;
    if (TexCoords.x != 0) {
       ambient  = light.ambient * vec3(texture(material.diffuse, TexCoords));
    }
    else {
        ambient = light.ambient * vec3(ourColor.x, ourColor.y, ourColor.z) ;
    }
        vec3 norm = normalize(Normal); // Diffuse
        vec3 lightDir = normalize(light.position - FragPos);
        float diff = max(dot(norm, lightDir), 0.0);

    if (TexCoords.x != 0){
        diffuse = light.diffuse * diff * vec3(texture(material.diffuse, TexCoords));
    } else {
        diffuse = light.diffuse * diff * vec3(ourColor.x, ourColor.y, ourColor.z) ;
    }
        vec3 viewDir = normalize(viewPos - FragPos);  // Specular
        vec3 reflectDir = reflect(-lightDir, norm);
        float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);
        vec3 specular = light.specular * (spec * material.specular);

        float distance    = length(light.position - FragPos);
        float attenuation = 1.0f / (light.constant + light.linear * distance +
                                    light.quadratic * (distance * distance));
        ambient  *= attenuation;
        diffuse  *= attenuation;
        specular *= attenuation;

        vec3 result = ambient + diffuse + specular;

        FragColor = vec4(result, ourColor.a);

        if (isBloom){
            BrightColor = vec4(result, ourColor.a);
        }
}
```

```
void FrameSystem::init()
{
    m_MainBuffer.init(FrameBuffer::MODE_READ_DRAW);
    m_MainBuffer.create();
    m_MainBuffer.bind();

    //новый кадр (FRAME)
    // Настройка фреймбуфера с плавающей запятой для рендеринга сцены
    // - Создание 2 цветных буферов с плавающей точкой (1 для обычного рендеринга, другие для значений порога яркости)

    for (GLuint i = 0; i < 2; i++)
    {
       m_ColorBuffer[i].create(Texture::Texture_2D);
       m_ColorBuffer[i].bind();
       m_ColorBuffer[i].setImage2D(Texture::RGB16, m_WidgetWidth, m_WidgetHeight,
                                   InputData::RGB, InputData::dtFloat, NULL, 0, 0);
       m_ColorBuffer[i].setFilter(Texture::Linear, Texture::Linear);
       m_ColorBuffer[i].setWrap(Texture::ClampToEdge, Texture::ClampToEdge);

        // подключение текстуры к фреймбуферу

       if (i == 0){
           m_MainBuffer.setTexture(FrameBuffer::COLOR0, m_ColorBuffer[i], 0);
       } else {
           m_MainBuffer.setTexture(FrameBuffer::COLOR1, m_ColorBuffer[i], 0);
       }
    }
    // - Создание и установка буфера глубины
    glGenRenderbuffers(1, &mRBO);
    glBindRenderbuffer(GL_RENDERBUFFER, mRBO);
    glRenderbufferStorage(GL_RENDERBUFFER,  GL_DEPTH24_STENCIL8, m_WidgetWidth, m_WidgetHeight);

    glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_STENCIL_ATTACHMENT, GL_RENDERBUFFER, mRBO);
    // - настройка OpenGL, какие цветные вложения будут использованы (из этого фреймбуфера) для рендеринга

    GLuint attachments[2] = { GL_COLOR_ATTACHMENT0, GL_COLOR_ATTACHMENT1 };
    glDrawBuffers(2, attachments);

    // - проверка состояния фреймбфуреа
    m_MainBuffer.checkStatus();

    glBindFramebuffer(GL_FRAMEBUFFER, 0);

    // Фреймбуфер для размытия
    for (GLuint i = 0; i < 2; i++)
    {
        m_GlowBuffer[i].init(FrameBuffer::MODE_READ_DRAW);
        m_GlowBuffer[i].create();
        m_GlowBuffer[i].bind();

        m_BloorColorBuffer[i].create(Texture::Texture_2D);
        m_BloorColorBuffer[i].bind();
        m_BloorColorBuffer[i].setImage2D(Texture::RGB16, m_WidgetWidth, m_WidgetHeight,
                                         InputData::RGB, InputData::dtFloat, NULL, 0, 0);
        m_BloorColorBuffer[i].setFilter(Texture::Linear, Texture::Linear);
        m_BloorColorBuffer[i].setWrap(Texture::ClampToEdge, Texture::ClampToEdge);

        // подключение текстуры к фреймбуферу

        m_GlowBuffer[i].setTexture(FrameBuffer::COLOR0, m_BloorColorBuffer[i], 0);
        m_GlowBuffer[i].checkStatus();
    }
}
```

```
void FrameSystem::makeGlow()
{
    glDisable(GL_MULTISAMPLE);
    glDisable(GL_BLEND);

    glDisable(GL_DEPTH_TEST);

    glBindFramebuffer(GL_FRAMEBUFFER, 0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // 1. Размытие ярких фрагментов с двухпроходным гауссовским размытием
    GLboolean horizontal = true, first_iteration = true;
    GLuint amount = 10;
    MainRenderModel->mShaders[BlurShader].bind();
    for (GLuint i = 0; i < amount; i++)
    {
        m_GlowBuffer[horizontal].bind();

        MainRenderModel->mShaders[BlurShader].setUniformValue("horizontal", horizontal);
        // Привязать текстуру другого фреймбуфера (или сцены, если первая итерация)
        glBindTexture(GL_TEXTURE_2D,
                      first_iteration ? m_ColorBuffer[1].id() : m_BloorColorBuffer[!horizontal].id());

        // ПОказывает 1x1 квадрат в Нормализованных координаты устройства
        RenderQuad();

        horizontal = !horizontal;
        if (first_iteration)
            first_iteration = false;
    }

    MainRenderModel->mShaders[BlurShader].unbind();
    glBindFramebuffer(GL_FRAMEBUFFER, 0);

    // 2.Теперь визуализируем цветной буфер с плавающей запятой в 2D quad
    // И цветовую карту tonemap HDR по умолчанию для цветовой гаммы фреймбуфера (зажатой)

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    MainRenderModel->mShaders[BloomFinalShader].bind();

    glActiveTexture(GL_TEXTURE0);
    m_ColorBuffer[0].bind();
    glActiveTexture(GL_TEXTURE1);
    m_BloorColorBuffer[!horizontal].bind();

    GLboolean bloom = true;

    MainRenderModel->mShaders[BloomFinalShader].setUniformValue("bloom", bloom);

    RenderQuad();

    MainRenderModel->mShaders[BloomFinalShader].unbind();

    m_MainBuffer.unbind();
    glBindBuffer(GL_ARRAY_BUFFER, 0);
}
```