

Informatyka praktyczna. Transformacje w OpenGL

Aleksander Denisiuk
Uniwersytet Warmińsko-Mazurski
Olsztyn, ul. Słoneczna 54
aleksander.denisiuk@uwm.edu.pl

Transformacje w OpenGL

Sześcian animowany

Implementacja

Najnowsza wersja tego dokumentu dostępna jest pod adresem

<http://wmii.uwm.edu.pl/~denisjuk/uwm>

Sześćcian animowany

- ❖ Sześćcian Animowany
- ❖ Tablice indeksów
- ❖ Zmienne uniform

Implementacja

Sześćcian animowany

Sześćcian Animowany

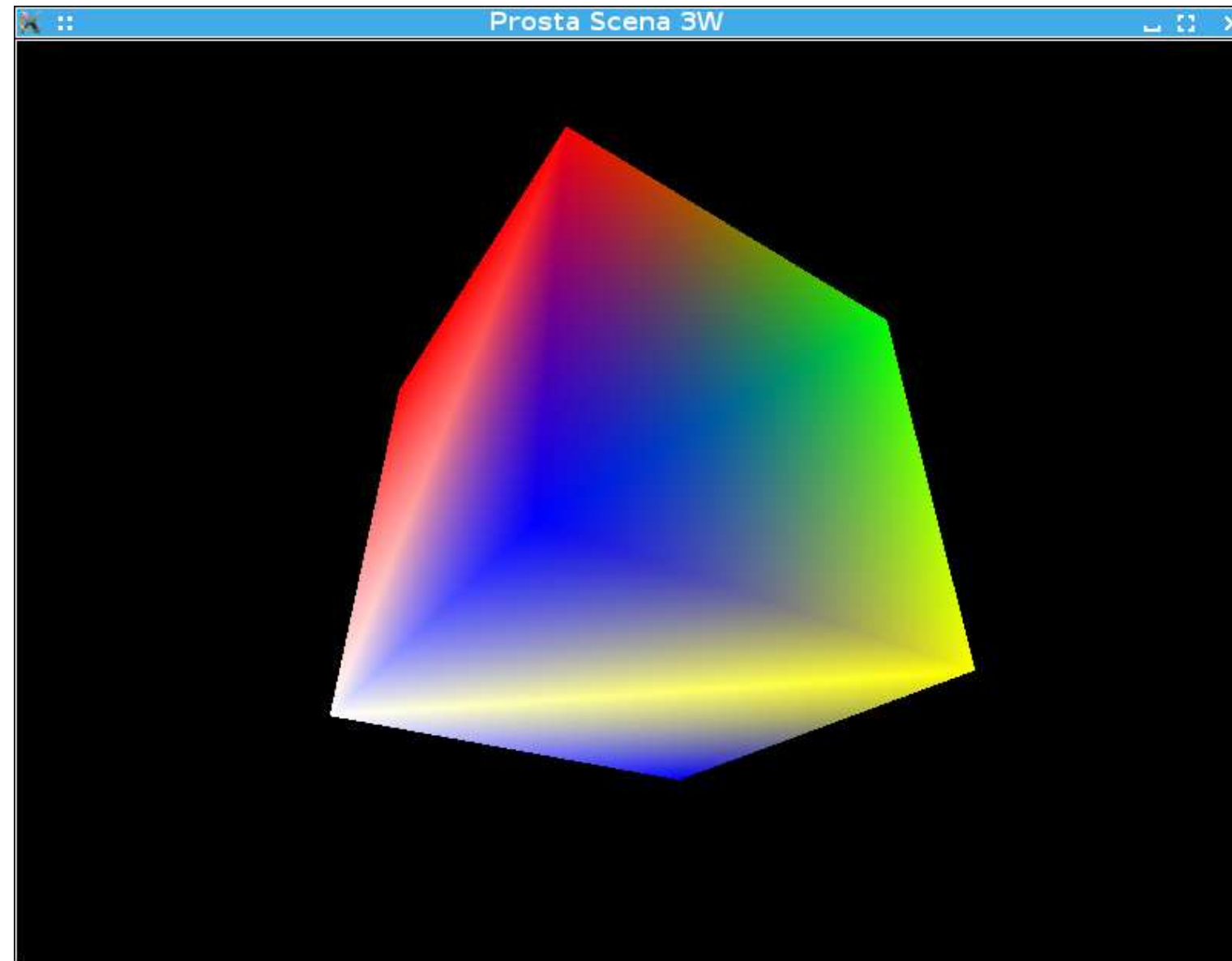
Sześćcian animowany

❖ Sześćcian Animowany

❖ Tablice indeksów

❖ Zmienne uniform

Implementacja



Zamodelujmy sześcian

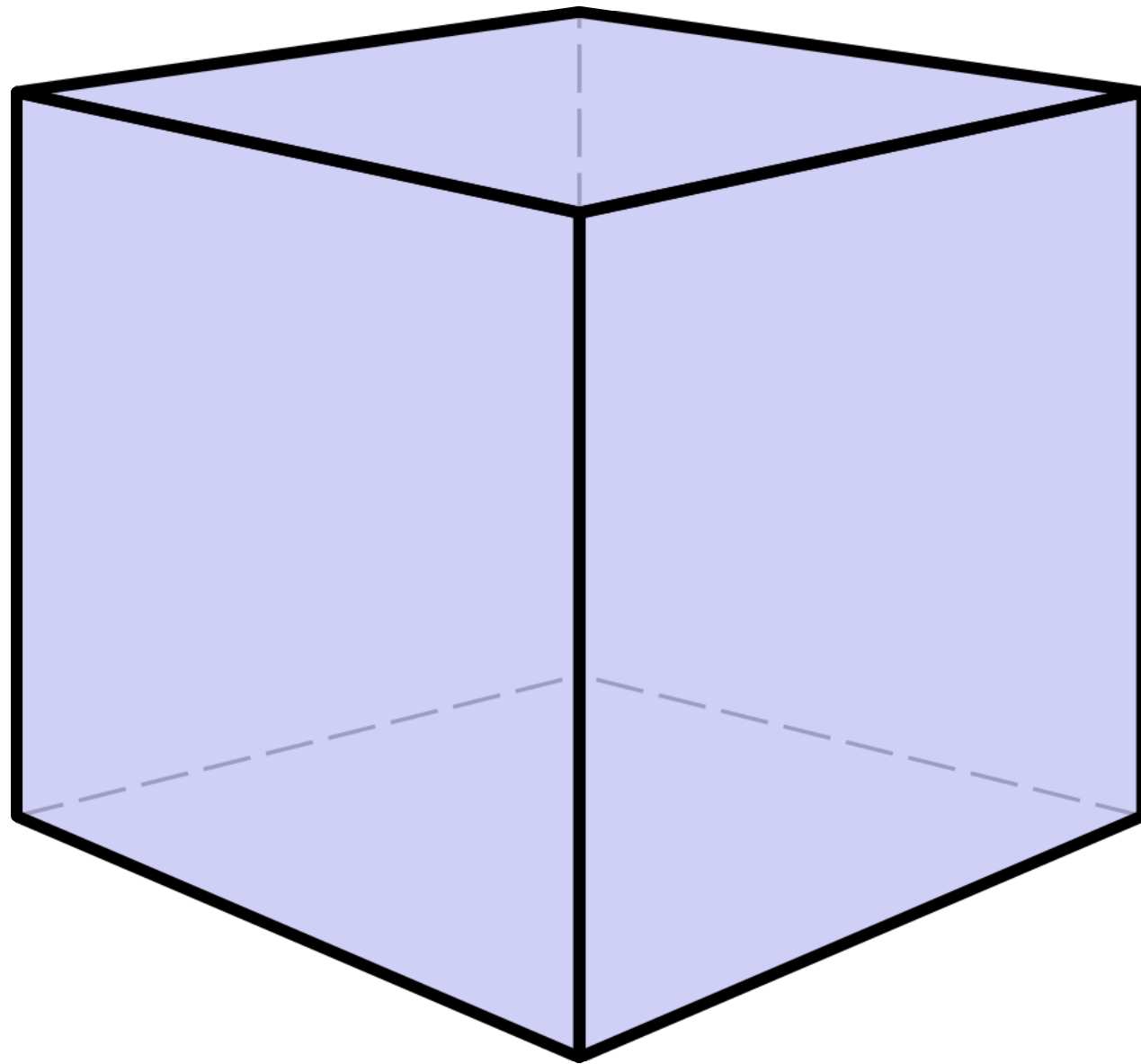
Sześcian animowany

❖ Sześcian Animowany

❖ **Tablice indeksów**

❖ Zmienne uniform

Implementacja



Podzielimy sześcian na trójkąty

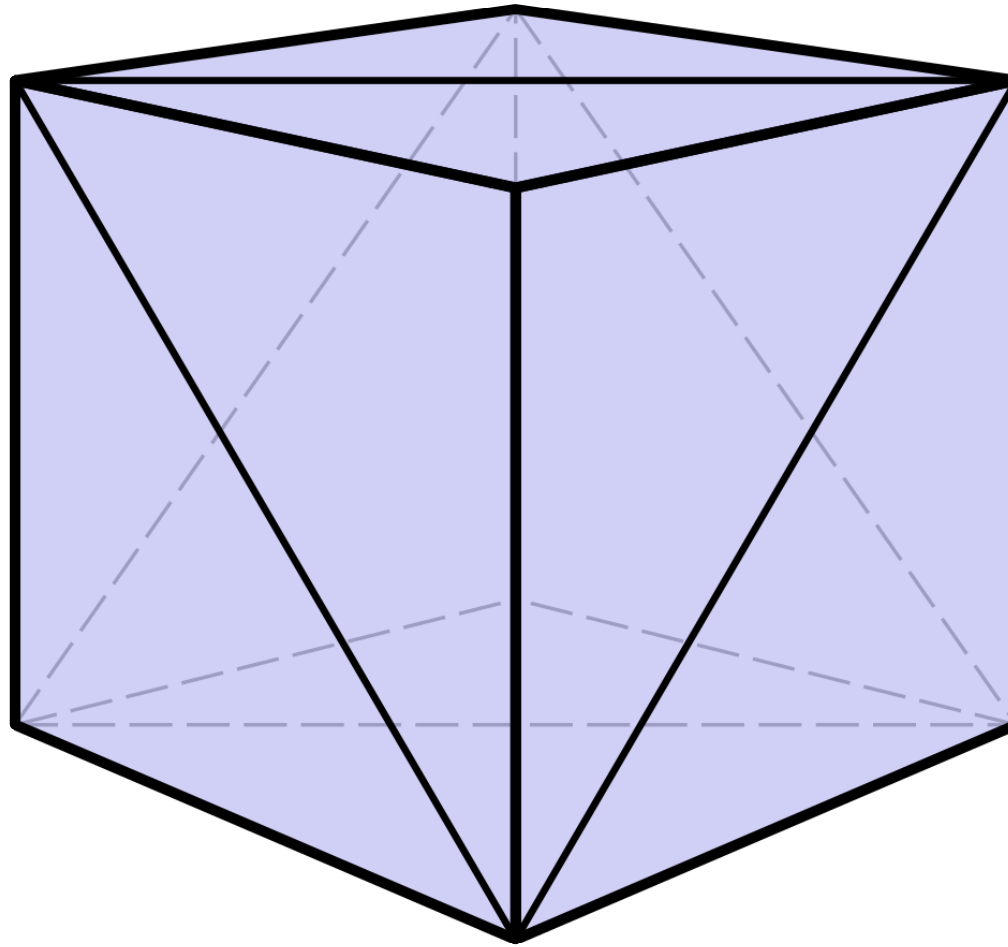
Sześcian animowany

❖ Sześcian Animowany

❖ Tablice indeksów

❖ Zmienne uniform

Implementacja



- Każdy wierzchołek należy do kilka trójkątów, więc jest wysyłany na GPU kilka razy

Rozwiązanie: tablice indeksów

Sześcian animowany

❖ Sześcian Animowany

❖ Tablice indeksów

❖ Zmienne uniform

Implementacja

- Każdy wierzchołek określa się (i wysyła się na GPU jeden raz)
 - ✦ wierzchołek może zawierać oprócz współrzędnych kolor, wektor normalny, współrzędne tekstury, etc
- Dodatkowo na GPU wysyła się tablicę indeksów, która określa geometrię
 - ✦ tablica zawiera numery indeksów wierzchołków, z których tworzy się poszczególne prymitywy graficzne
 - ✦ indeksy — liczby całkowite (`GLbyte`, `GLshort`, `GLint`, `GLubyte`, `GLushort`, `GLuint`)

VBO dla tablicy indeksów

Sześcian animowany

❖ Sześcian Animowany

❖ Tablice indeksów

❖ Zmienne uniform

Implementacja

- Zmienna `index_buffer` ma typ `GLuint`
- Zmienna `kIndices` jest tablicą indeksów

```
glGenBuffers(1, &index_buffer);  
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,  
             index_buffer);  
glBufferData(GL_ELEMENT_ARRAY_BUFFER,  
             sizeof(kIndices),  
             kIndices,  
             GL_STATIC_DRAW);
```

- Tablica indeksów jest związana z bieżącym VAO
 - ✦ Po (przed) `glBindVertexArray(0)` nie robimy `glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0)`
 - ✦ Przy renderowaniu wystarczy `glBindVertexArray`

Renderowanie za pomocą tablicy indeksów

Sześcian animowany

❖ Sześcian Animowany

❖ Tablice indeksów

❖ Zmienne uniform

Implementacja

```
void glDrawElements (  
    GLenum mode,  
    GLsizei count,  
    GLenum type,  
    const GLvoid * indices);
```

Przykład dla sześcianu. Struktura

Sześcian animowany

❖ Sześcian Animowany

❖ Tablice indeksów

❖ Zmienne uniform

Implementacja

```
typedef struct {  
    float position[4];  
    float color[4];  
} ColorVertex;
```

Przykład dla sześcianu. Wierzchołki

Sześcian animowany

❖ Sześcian Animowany

❖ Tablice indeksów

❖ Zmienne uniform

Implementacja

```
const ColorVertex kVertices[8] =
{
    { { -.5f, -.5f, .5f, 1.0f }, { 0, 0, 1, 1 } },
    { { -.5f, .5f, .5f, 1.0f }, { 1, 0, 0, 1 } },
    { { .5f, .5f, .5f, 1.0f }, { 0, 1, 0, 1 } },
    { { .5f, -.5f, .5f, 1.0f }, { 1, 1, 0, 1 } },
    { { -.5f, -.5f, -.5f, 1.0f }, { 1, 1, 1, 1 } },
    { { -.5f, .5f, -.5f, 1.0f }, { 1, 0, 0, 1 } },
    { { .5f, .5f, -.5f, 1.0f }, { 1, 0, 1, 1 } },
    { { .5f, -.5f, -.5f, 1.0f }, { 0, 0, 0, 1 } }
};
```

Przykład dla sześcianu. Indeksy

Sześcián animowany

❖ Sześcián Animowany

❖ Tablice indeksów

❖ Zmienne uniform

Implementacja

```
const GLuint kIndices[36] =  
{  
    0, 1, 2,    0, 2, 3,  
    4, 0, 3,    4, 3, 7,  
    4, 5, 1,    4, 1, 0,  
    3, 2, 6,    3, 6, 7,  
    1, 5, 6,    1, 6, 2,  
    7, 6, 5,    7, 5, 4  
};
```

Przykład dla sześcianu. VBO dla wierzchołków

Sześcian animowany

❖ Sześcian Animowany

❖ Tablice indeksów

❖ Zmienne uniform

Implementacja

```
glGenBuffers(1, &vertex_buffer);
glBindBuffer(GL_ARRAY_BUFFER, vertex_buffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(kVertices),
             kVertices, GL_STATIC_DRAW);
glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE,
                     sizeof(kVertices[0]), (GLvoid*) 0);
glEnableVertexAttribArray(0);
glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE,
                     sizeof(kVertices[0]),
                     (GLvoid*) sizeof(kVertices[0].position));
glEnableVertexAttribArray(1);
```

Przykład dla sześcianu. VBO dla indeksów

Sześcian animowany

❖ Sześcian Animowany

❖ Tablice indeksów

❖ Zmienne uniform

Implementacja

```
glGenBuffers(1, &index_buffer);  
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,  
             index_buffer);  
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(kIndices),  
             kIndices, GL_STATIC_DRAW);
```

Przykład dla sześcianu. Wyświetlenie

Sześcian animowany

❖ Sześcian Animowany

❖ Tablice indeksów

❖ Zmienne uniform

Implementacja

```
glUseProgram(program);  
glBindVertexArray(vao);
```

```
glDrawElements(GL_TRIANGLES, 36,  
               GL_UNSIGNED_INT, 0);
```

```
glBindVertexArray(0);  
glUseProgram(0);
```

Zmienne uniform

Sześcian animowany

❖ Sześcian Animowany

❖ Tablice indeksów

❖ Zmienne uniform

Implementacja

- Do animacji sześcianu używamy transformacji geometrycznych
 - ◆ Układ lokalny (*object space*)
 - ◆ Układ globalny (*world space*)
 - ◆ Układ obserwatora (*eye space*)
- Trzy macierze:
 - ◆ *model matrix*
 - ◆ *view matrix*
 - ◆ *projection matrix*
- Obliczamy odpowiednie macierze w programie
- Przekazujemy macierze do GPU (do shadera, do zmiennych *uniform*)
- W shaderze (którym?) obliczamy współrzędne po transformacjach

Deklaracja w Shaderze

Sześcian animowany

❖ Sześcian Animowany

❖ Tablice indeksów

❖ Zmienne uniform

Implementacja

```
uniform mat4 model_matrix;  
uniform mat4 view_matrix;  
uniform mat4 projection_matrix;
```

W programie

Sześcian animowany

❖ Sześcian Animowany

❖ Tablice indeksów

❖ Zmienne uniform

Implementacja

- Znaleźć adres zmiennej uniform w shaderze

```
glUseProgram(program);  
location = glGetUniformLocation(program, var_name);
```

- ◆ zmienna `location` powinna mieć typ `GLint`
- ◆ `var_name` jest tekstem (`const char*`), który zawiera dokładnie taką nazwę zmiennej uniform, jaka jest w shaderze
- ◆ jeżeli zmiennej uniform nie znaleziono, w `location` zostanie zapisano `-1`

- przykładowo:

```
location=glGetUniformLocation(program, "model_matrix");
```

Wysyłanie danych

Sześcian animowany

❖ Sześcian Animowany

❖ Tablice indeksów

❖ Zmienne uniform

Implementacja

- Do wysyłania danych do zmiennej uniform służą funkcje

`glUniform*`

- Na przykład:

- ◆ liczba całkowita

```
glUniform1i(GLint location, GLint v0);
```

- ◆ czterowymiarowy wektor `GLfloat` (tablicę wektorów):

```
glUniform4fv(GLint location, GLsizei count,  
             const GLfloat *value);
```

- ◆ macierz 4×4 :

```
glUniformMatrix4fv(GLint location,  
                   GLsizei count, GLboolean transpose,  
                   const GLfloat *value);
```

- ◆ etc

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ `vertices.h`
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ `matma`
- ❖ Algebra macierzy

Implementacja

Shader Wierzchołków

Sześcian animowany

Implementacja

❖ Shadery

❖ Klasy C++

❖ vertices.h

❖ Hierarchia modeli

❖ Cube

❖ Program podstawowy

❖ CameraProgram

❖ ModelProgram

❖ Window

❖ matma

❖ Algebra macierzy

```
#version 430 core

layout(location=0) in vec4 in_position;
layout(location=1) in vec4 in_color;

out vec4 frag_color;

uniform mat4 model_matrix;
uniform mat4 view_matrix;
uniform mat4 projection_matrix;

void main(void) {
    gl_Position = (projection_matrix
                  * view_matrix * model_matrix)
                  * in_position;
    frag_color = in_color;
}
```

Shader Fragmentów

Sześcian animowany

Implementacja

❖ Shadery

❖ Klasy C++

❖ vertices.h

❖ Hierarchia modeli

❖ Cube

❖ Program podstawowy

❖ CameraProgram

❖ ModelProgram

❖ Window

❖ matma

❖ Algebra macierzy

```
#version 430 core
```

```
layout (location = 0) out vec4 color;
```

```
in vec4 frag_color;
```

```
void main(void) {  
    color = frag_color;  
}
```

Klasy C++

Sześcian animowany

Implementacja

❖ Shadery

❖ Klasy C++

❖ `vertices.h`

❖ Hierarchia modeli

❖ Cube

❖ Program podstawowy

❖ CameraProgram

❖ ModelProgram

❖ Window

❖ `matma`

❖ Algebra macierzy

- Window — odpowiada za kontekst
- `vertices.h` — deklaracje struktur dla wieszchołków
- Cube, IndexModel, MovableModel — hierarchia klas modeli
- ModleProgram, CameraProgram, BaseProgram — hierarchia klas programów
- `matma.h` — algebra macierzy
- `glerror.h` — definicje, związane z debugowaniem (bez zmian)
- `main.c` — program główny

Struktura dla wierzchołków

Sześcian animowany

Implementacja

❖ Shadery

❖ Klasy C++

❖ **vertices.h**

❖ Hierarchia modeli

❖ Cube

❖ Program podstawowy

❖ CameraProgram

❖ ModelProgram

❖ Window

❖ matma

❖ Algebra macierzy

```
typedef struct ColorVertex{  
    GLfloat position[4];  
    GLfloat color[4];  
} ColorVertex;
```

Model z tablicą indeksów

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h

❖ Hierarchia modeli

- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

IndexModel

```
# vao_ : GLuint
# vertex_buffer_ : GLuint
# index_buffer_ : GLuint

+ ~IndexModel ()
```

- Każdy obiekt będzie miał własną funkcję Initialize()

```
IndexModel::~IndexModel () {
    glDisableVertexAttribArray(1);
    glDisableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
    glDeleteBuffers(1, &index_buffer_);
    glDeleteBuffers(1, &vertex_buffer_);
    glBindVertexArray(0);
    glDeleteVertexArrays(1, &vao_);
}
```

Model przemieszczalny

Sześcian animowany

Implementacja

❖ Shadery

❖ Klasy C++

❖ `vertices.h`

❖ Hierarchia modeli

❖ Cube

❖ Program podstawowy

❖ CameraProgram

❖ ModelProgram

❖ Window

❖ `matma`

❖ Algebra macierzy

MovableModel

```
# model_matrix_ : Mat4
```

- ❖ Shadery
- ❖ Klasy C++
- ❖ `vertices.h`
- ❖ Hierarchia modeli
- ❖ **Cube**
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ `matma`
- ❖ Algebra macierzy

IndexModel

MovableModel

Cube

- `velocity_x_` : float
- `velocity_y_` : float
- `animated_` : bool

- + `Cube(init_velocity_x=45, init_velocity_y=45)`
- + `Initialize ()`
- + `Move (delta_t: float)`
- + `SpeedUp ()`
- + `SlowDown ()`
- + `ToggleAnimated ()`
- + `IsAnimated (): bool`
- + `SetVelocityX (velocity : float)`
- + `SetVelocityY (velocity : float)`
- + `Draw ( program : ModelProgram )`

Inicjalizacja

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ `vertices.h`
- ❖ Hierarchia modeli
- ❖ **Cube**
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ `matma`
- ❖ Algebra macierzy

- Wierzchołki, VAO, VBO — jak wyżej

Animacja

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ **Cube**
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
void Cube::Move(float delta_t) {  
    if (!animated_) return;  
    GLfloat angle_x = delta_t * velocity_x_;  
    model_matrix_.RotateAboutX(angle_x);  
    GLfloat angle_y = delta_t * velocity_y_;  
    model_matrix_.RotateAboutY(angle_y);  
}
```

Ustawienie animacji

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ **Cube**
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
void Cube::SpeedUpX() {  
    velocity_x_ *= 1.09544511501;  
}
```

```
void Cube::SlowDownX() {  
    velocity_x_ /= 1.09544511501;  
}
```

```
void Cube::ToggleAnimated() {  
    animated_ = ! animated_;  
}
```

Renderowanie

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ **Cube**
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
void Cube::Draw(const ModelProgram & program) {  
  
    glUseProgram(program);  
    glBindVertexArray(vao_);  
  
    program.SetModelMatrix(model_matrix_);  
  
    glDrawElements(GL_TRIANGLES, 36,  
                   GL_UNSIGNED_INT, 0);  
  
    glBindVertexArray(0);  
    glUseProgram(0);  
  
}
```

Program podstawowy

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ `vertices.h`
- ❖ Hierarchia modeli
- ❖ Cube

❖ Program podstawowy

- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ `matma`
- ❖ Algebra macierzy

BaseProgram

```
# program_ : GLuint
- vertex_shader_ : GLuint
- fragment_shader_ : GLuint

+ operator GLuint() const
+ ~BaseProgram ()
+ Initialize (vertex_shader_file, fragment_shader_file)
- LoadAndCompileShaderOrDie ( source_file, type )
- LinkProgramOrDie ( vertex_shader, fragment_shader )
# GLint GetUniformLocationOrDie(const char *);
```

Deklaracja

Sześcian animowany

Implementacja

❖ Shadery

❖ Klasy C++

❖ vertices.h

❖ Hierarchia modeli

❖ Cube

❖ Program podstawowy

❖ CameraProgram

❖ ModelProgram

❖ Window

❖ matma

❖ Algebra macierzy

```
class BaseProgram{
public:
    void Initialize(const char* vertex_shader_file,
                   const char* fragment_shader_file);
    operator GLuint() const{return program_;}
    ~BaseProgram();
protected:
    GLuint program_;

    .....
```

Deklaracja, cd

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
.....  
private:  
    GLuint vertex_shader_;  
    GLuint fragment_shader_;  
    GLuint LoadAndCompileShaderOrDie (  
        const char* source_file,  
        GLenum type);  
    GLuint LinkProgramOrDie (  
        GLint vertex_shader,  
        GLint fragment_shader);  
protected:  
    GLint GetUniformLocationOrDie (const char *);  
};
```

Inicjalizacja

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ `vertices.h`
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ `matma`
- ❖ Algebra macierzy

```
vertex_shader_ = LoadAndCompileShaderOrDie(  
    vertex_shader_file, GL_VERTEX_SHADER);  
fragment_shader_ = LoadAndCompileShaderOrDie(  
    fragment_shader_file, GL_FRAGMENT_SHADER);  
program_ = LinkProgramOrDie(vertex_shader_,  
    fragment_shader_);
```

LoadAndCompileShaderOrDie

Sześcian animowany

Implementacja

❖ Shadery

❖ Klasy C++

❖ vertices.h

❖ Hierarchia modeli

❖ Cube

❖ Program podstawowy

❖ CameraProgram

❖ ModelProgram

❖ Window

❖ matma

❖ Algebra macierzy

```
GLuint Program::LoadAndCompileShaderOrDie(  
    const char * source_file, GLenum type){  
    int file_size;  
    char * shader_code;  
    GLuint shader=glCreateShader(type);  
    ifstream file (source_file, ios::in|ios::ate);  
    if (file.is_open()) {  
        file_size = file.tellg();  
        shader_code = new char [file_size+1];  
        file.seekg (0, ios::beg);  
        file.read (shader_code, file_size);  
        shader_code[file_size]='\0';  
        file.close();  
        glShaderSource(shader, 1,  
                        (const GLchar**) &shader_code, NULL);  
        glCompileShader(shader);  
        delete[] shader_code;  
    }  
}
```

LoadAndCompileShaderOrDie, cd

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube

❖ Program podstawowy

- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
else{ //file was not opened
    cerr<<"Could not open the file "<<source_file<<endl;
    glfwTerminate();
    exit( EXIT_FAILURE );
}
```

- Dalej sprawdzanie wyniku kompilacji — jak wcześniej

GetUniformLocationOrDie

Sześcian animowany

Implementacja

❖ Shadery

❖ Klasy C++

❖ vertices.h

❖ Hierarchia modeli

❖ Cube

❖ Program podstawowy

❖ CameraProgram

❖ ModelProgram

❖ Window

❖ matma

❖ Algebra macierzy

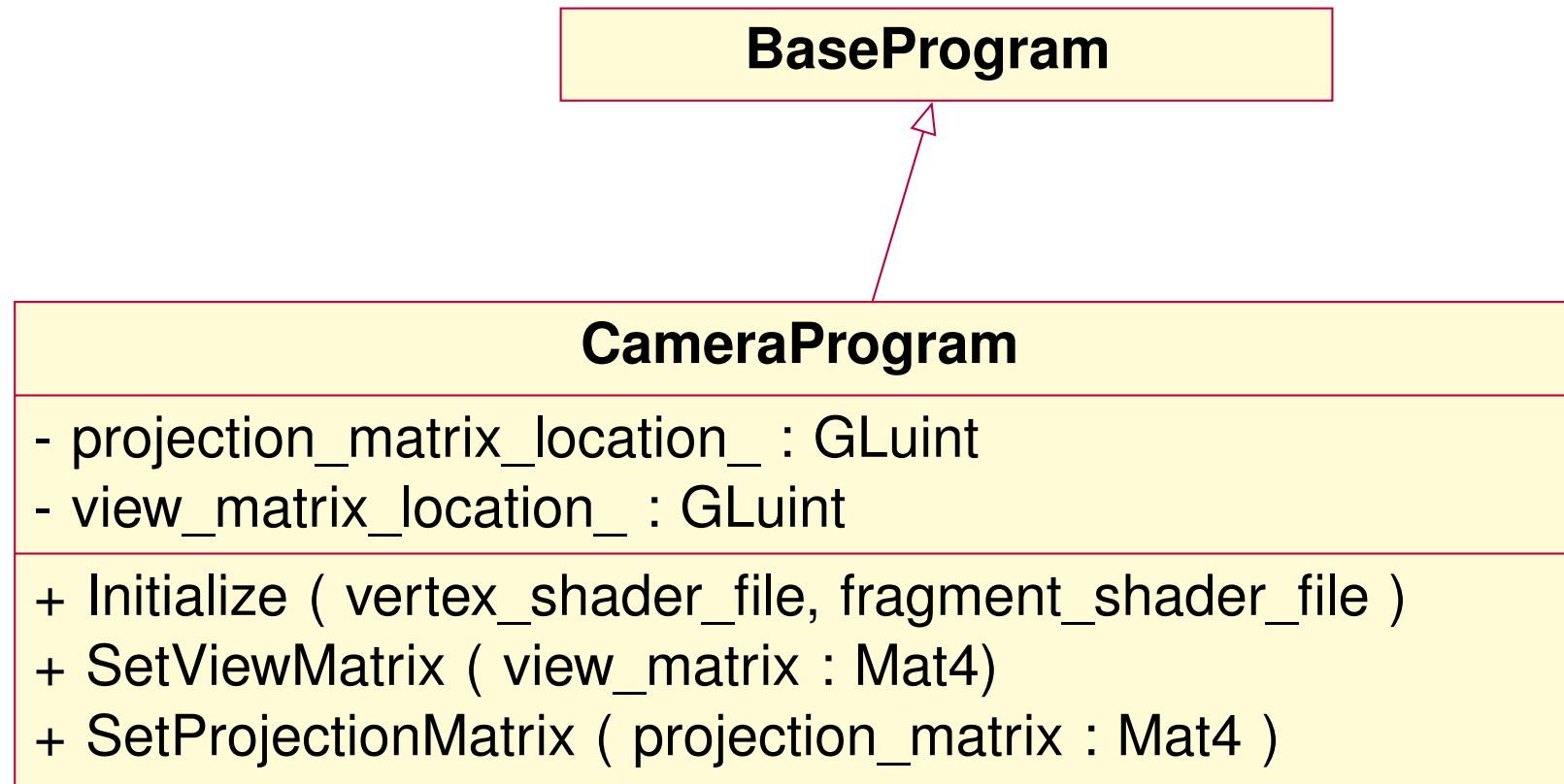
```
GLint BaseProgram::GetUniformLocationOrDie(  
    const char* var_name) {  
    GLint location=-1;  
    location = glGetUniformLocation(program_, var_name);  
    if (location < 0) {  
        cerr << "ERROR: cannot find uniform location "  
              << var_name << endl;  
        glfwTerminate();  
        exit( EXIT_FAILURE );  
    }  
    return location;  
}
```

CameraProgram

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ `vertices.h`
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ **CameraProgram**
- ❖ ModelProgram
- ❖ Window
- ❖ `matma`
- ❖ Algebra macierzy



Deklaracja

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ **CameraProgram**
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
class CameraProgram : public BaseProgram{
public:
    void Initialize(
        const char* vertex_shader_file,
        const char* fragment_shader_file);
    void SetViewMatrix(const Mat4 &) const;
    void SetProjectionMatrix(const Mat4 &) const;
private:
    GLuint projection_matrix_location_;
    GLuint view_matrix_location_;
};
```

Inicjalizacja

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ **CameraProgram**
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
void CameraProgram::Initialize(  
    const char *vertex_shader_file,  
    const char *fragment_shader_file) {  
    BaseProgram::Initialize(  
        vertex_shader_file,  
        fragment_shader_file);  
    projection_matrix_location_  
        = glGetUniformLocationOrDie("projection_matrix");  
    view_matrix_location_  
        = glGetUniformLocationOrDie("view_matrix");  
}
```

Zmienne uniform

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ **CameraProgram**
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
void CameraProgram::SetProjectionMatrix(  
    const Mat4 & matrix) {  
    glUniformMatrix4fv(projection_matrix_location_,  
        1, GL_FALSE, matrix);  
}
```

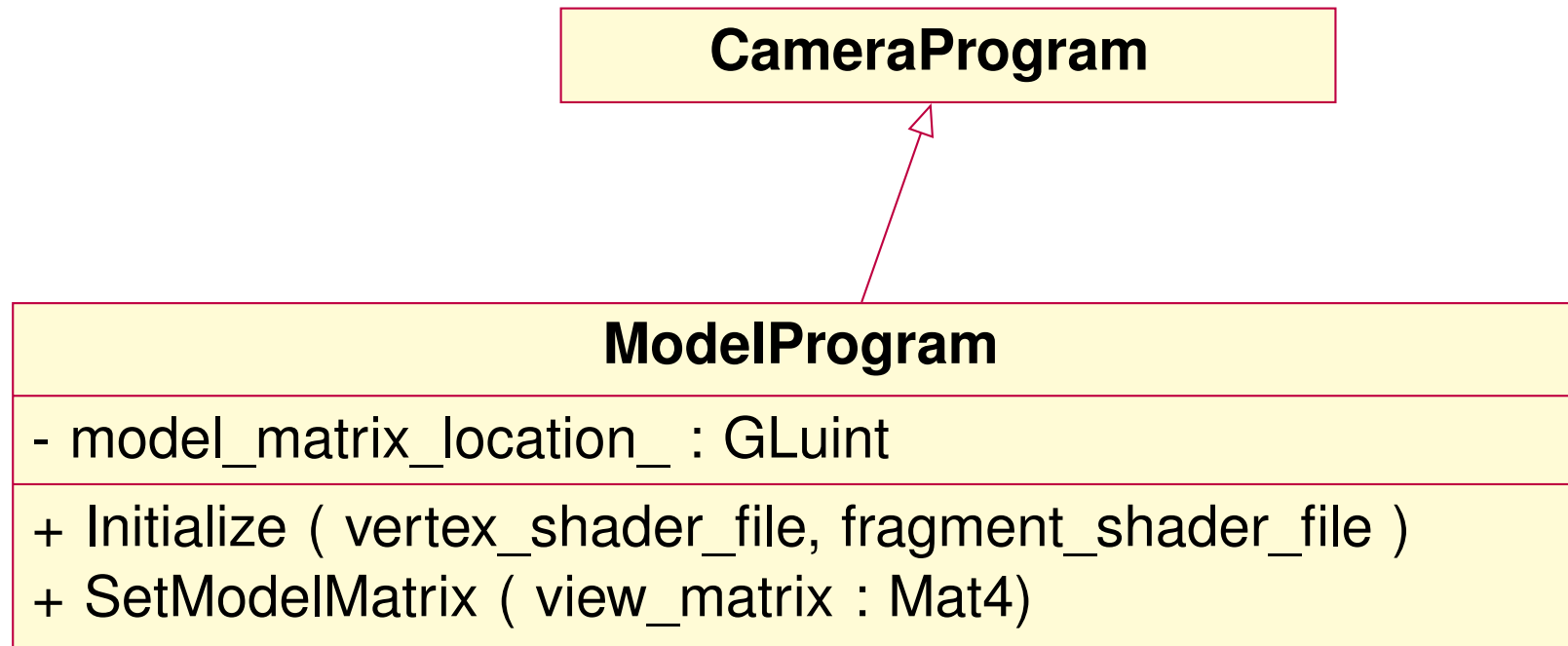
```
void CameraProgram::SetViewMatrix(const Mat4 & matrix) {  
    glUniformMatrix4fv(view_matrix_location_,  
        1, GL_FALSE, matrix);  
}
```

ModelProgram

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ `vertices.h`
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ **ModelProgram**
- ❖ Window
- ❖ `matma`
- ❖ Algebra macierzy



Inicjalizacja

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ **ModelProgram**
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
void ModelProgram::Initialize(  
    const char *vertex_shader_file,  
    const char *fragment_shader_file) {  
    CameraProgram::Initialize(  
        vertex_shader_file,  
        fragment_shader_file);  
    model_matrix_location_  
        = glGetUniformLocation("model_matrix");  
}
```

Zmienne uniform

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ **ModelProgram**
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
void ModleProgram::SetModelMatrix(const Mat4 & matrix) {  
    glUniformMatrix4fv(model_matrix_location_,  
                        1, GL_FALSE, matrix);  
}
```

Zmiany w klasie Window

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

- nowe dane

private:

```
Cube cube_;  
clock_t last_time_;  
Mat4 view_matrix_;  
Mat4 projection_matrix_;
```

- dwie nowe metody prywatne:

```
void SetViewMatrix() const;  
void SetProjectionMatrix() const;
```

- dwie stałe w pliku window.cpp:

```
const char* kVertexShader="SimpleShader.vertex.glsl";  
const char* kFragmentShader  
="SimpleShader.fragment.glsl";
```

Zmiany w inicjalizacji

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

- w konstruktorze:

```
last_time_ = 0;
```

- w procedurze Initialize:

```
InitModels();
```

```
InitPrograms();
```

```
view_matrix_.Translate(0, 0, -2);
```

```
SetViewMatrix();
```

```
projection_matrix_ = Mat4::CreateProjectionMatrix(  
    60, (float)width_/(float)height_, 0.1f, 100.0f);  
SetProjectionMatrix();
```

```
glEnable(GL_DEPTH_TEST);
```

```
glDepthFunc(GL_LESS);
```

Renderowanie

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
void Window::Run(void) {
    while (!glfwWindowShouldClose(window_)) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        clock_t now = clock();
        if (last_time_ == 0) last_time_ = now;
        cube_.Move( (float)(now - last_time_)
                    / CLOCKS_PER_SEC );
        last_time_ = now;

        cube_.Draw(program_);
        glfwSwapBuffers(window_);
        glfwPollEvents();
    }
}
```

Resize

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
void Window::Resize(int new_width, int new_height) {
    width_ = new_width;
    height_ = new_height;
    projection_matrix_ = Mat4::CreateProjectionMatrix(
        60, (float)width_ / (float)height_, 0.1f, 100.0f);
    SetProjectionMatrix();
    glViewport(0, 0, width_, height_);
}
```

Macierze kamery

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
void Window::SetViewMatrix() const{  
    glUseProgram(program_);  
    program_.SetViewMatrix(view_matrix_);  
}
```

```
void Window::SetProjectionMatrix() const{  
    glUseProgram(program_);  
    program_.SetProjectionMatrix(projection_matrix_);  
}
```

KeyEvent, naciśnięcie

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
void Window::KeyEvent(int key, int /*scancode*/,
                      int action, int /*mods*/) {
    if(action == GLFW_PRESS) {
        switch (key) {
            case GLFW_KEY_ESCAPE:
                glfwSetWindowShouldClose(window_, GLFW_TRUE);
                break;
            case GLFW_KEY_SPACE:
                cube_.ToggleAnimated();
                break;
            case GLFW_KEY_LEFT:
                cube_.SlowDownY();
                break;
            case GLFW_KEY_RIGHT:
                cube_.SpeedUpY();
                break;
```

.....

KeyEvent, przytrzymywanie

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
.....
else if(action == GLFW_REPEAT) {
    switch (key) {
        case GLFW_KEY_LEFT:
            cube_.SlowDownY();
        break;
        case GLFW_KEY_RIGHT:
            cube_.SpeedUpY();
        break;
        case GLFW_KEY_DOWN:
            cube_.SlowDownX();
        break;
        case GLFW_KEY_UP:
            cube_.SpeedUpX();
        break;
    }
}
```

Deklaracja π

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ `vertices.h`
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ **matma**
- ❖ Algebra macierzy

```
#ifndef M_PI  
    #define M_PI 3.14159265  
#endif
```

Algebra macierzy

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
class Mat4{
public:
    Mat4(); // Unit matrix
    operator const float* () const{return matrix_;}
    static Mat4 CreatePerspectiveProjectionMatrix(
        float fovy, float aspect_ratio,
        float near_plane, float far_plane);
    void RotateAboutX(float angle); //gedrees
    void RotateAboutY(float angle); //gedrees
    void RotateAboutZ(float angle); //gedrees
    void Scale(float x_scale, float y_scale,
               float z_scale);
    void Translate(float delta_x, float delta_y,
                  float delta_z);

    void SetUnitMatrix();
    void Log();
```

Algebra macierzy

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ `vertices.h`
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ `matma`
- ❖ Algebra macierzy

```
private:
    float matrix_[16]; // column-major
    void MultiplyBy(const Mat4 &);
    explicit Mat4(float);
};
```

- Uwaga: w pamięci GPU macierze umieszczone są wzdłuż kolumn
- Dwa podejścia:
 - ✦ w programie macierze umieszczamy w zwykły sposób, przy wysyłaniu na GPU transponujemy
 - ✦ w programie macierze umieszczamy wzdłuż kolumn, przy wysyłaniu na GPU nie transponujemy
- wybrane pierwsze

Konstruktory i jednostkowanie

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
Mat4::Mat4() { // Unit matrix
    for (int i=0; i<16; i++) matrix_[i]=0;
    matrix_[0]=matrix_[5]=matrix_[10]=matrix_[15]=1;
}
```

```
Mat4::Mat4(float val) {
    for (int i=0; i<16; i++) {
        matrix_[i]=val;
    }
}
```

```
void Mat4::SetUnitMatrix() { // Unit matrix
    for (int i=0; i<16; i++) matrix_[i]=0;
    matrix_[0]=matrix_[5]=matrix_[10]=matrix_[15]=1;
}
```

Mnożenie

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
void Mat4::MultiplyBy(const Mat4 & m2) {
    float new_matrix[16];
    unsigned int row, column, row_offset;
    for (row = 0, row_offset = row * 4;
         row < 4; ++row, row_offset = row * 4)
        for (column = 0; column < 4; ++column)
            new_matrix[row_offset + column] =
                (matrix_[row_offset + 0] * m2.matrix_[column + 0])
                + (matrix_[row_offset + 1] * m2.matrix_[column + 4])
                + (matrix_[row_offset + 2] * m2.matrix_[column + 8])
                + (matrix_[row_offset + 3] * m2.matrix_[column + 12]);
    for (int i=0; i<16; i++) matrix_[i]=new_matrix[i];
}
```

Skalowanie

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
void Mat4::Scale(float x, float y, float z) {  
    Mat4 scale;  
  
    scale.matrix_[0] = x;  
    scale.matrix_[5] = y;  
    scale.matrix_[10] = z;  
  
    MultiplyBy(scale);  
}
```

Translacja

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
void Mat4::Translate(float x, float y, float z) {  
    Mat4 translate;  
  
    translate.matrix_[12] = x;  
    translate.matrix_[13] = y;  
    translate.matrix_[14] = z;  
  
    MultiplyBy(translate);  
}
```

Obrót dookoła O_x

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
void Mat4::RotateAboutX(float degrees) {  
    Mat4 rotation;  
    float radians=degrees*M_PI/180.0f;  
    float sine = (float)sin(radians);  
    float cosine = (float)cos(radians);  
  
    rotation.matrix_[5] = cosine;  
    rotation.matrix_[6] = sine;  
    rotation.matrix_[9] = -sine;  
    rotation.matrix_[10] = cosine;  
  
    MultiplyBy(rotation);  
}
```

Obrót dookoła O_y

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
void Mat4::RotateAboutY(float degrees) {  
    Mat4 rotation;  
    float radians=degrees*M_PI/180.0f;  
    float sine = (float)sin(radians);  
    float cosine = (float)cos(radians);  
  
    rotation.matrix_[0] = cosine;  
    rotation.matrix_[2] = -sine;  
    rotation.matrix_[8] = sine;  
    rotation.matrix_[10] = cosine;  
  
    MultiplyBy(rotation);  
}
```

Obrót dookoła O_z

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

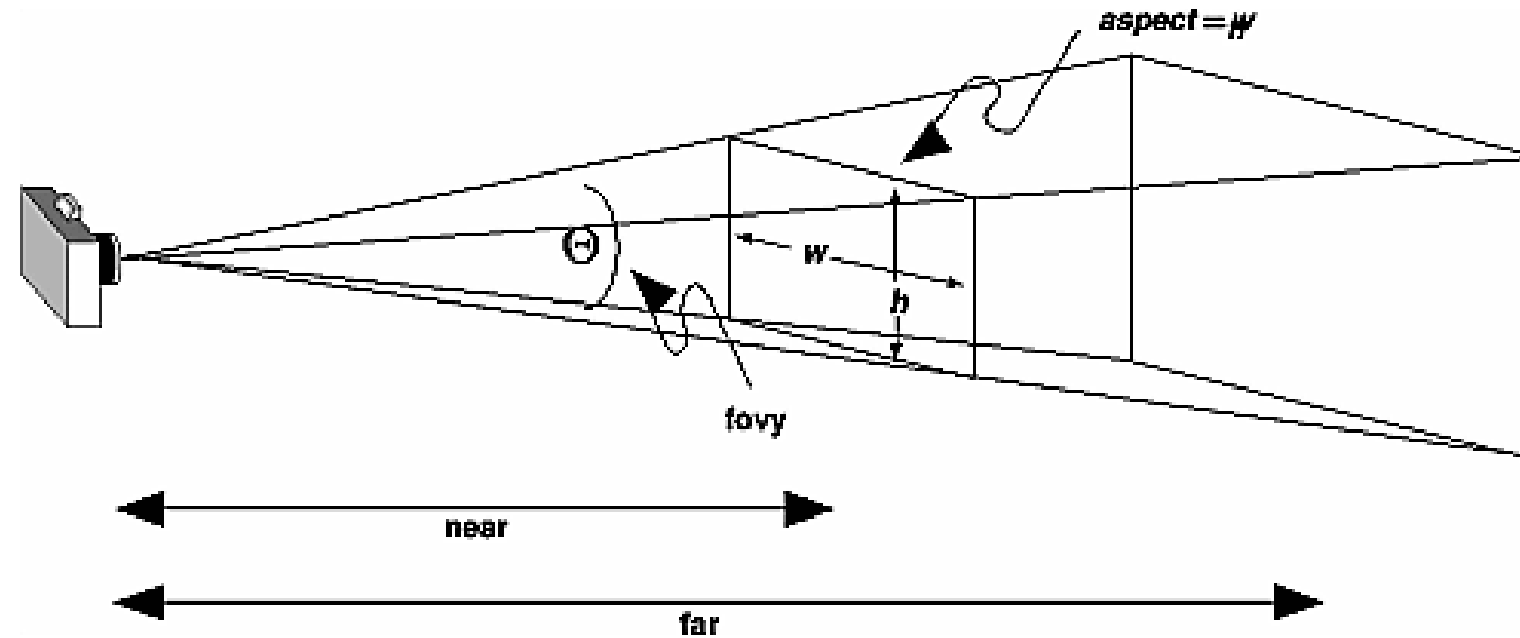
```
void Mat4::RotateAboutZ(float degrees) {  
    Mat4 rotation;  
    float radians=degrees*M_PI/180.0f;  
    float sine = (float)sin(radians);  
    float cosine = (float)cos(radians);  
  
    rotation.matrix_[0] = cosine;  
    rotation.matrix_[1] = sine;  
    rotation.matrix_[4] = -sine;  
    rotation.matrix_[5] = cosine;  
  
    MultiplyBy(rotation);  
}
```

Macierz rzutowania

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ `vertices.h`
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ `matma`
- ❖ Algebra macierzy



$$\begin{bmatrix} xScale & 0 & 0 & 0 \\ 0 & yScale & 0 & 0 \\ 0 & 0 & -\frac{zFar+zNear}{zFar-zNear} & -\frac{2 \cdot zNear \cdot zFar}{zFar-zNear} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

gdzie $yScale = \cot(\frac{fovy}{2})$ oraz $xScale = \frac{f}{aspect}$.

Implementacja

Sześcian animowany

Implementacja

- ❖ Shadery
- ❖ Klasy C++
- ❖ vertices.h
- ❖ Hierarchia modeli
- ❖ Cube
- ❖ Program podstawowy
- ❖ CameraProgram
- ❖ ModelProgram
- ❖ Window
- ❖ matma
- ❖ Algebra macierzy

```
Mat4 Mat4::CreatePerspectiveProjectionMatrix(  
    float fovy, float aspect_ratio,  
    float near_plane, float far_plane) {  
  
    Mat4 out(0);  
    float y_scale = 1.0/tan(fovy * M_PI / 360.0 );  
    float x_scale = y_scale / aspect_ratio;  
    float frustum_length = far_plane - near_plane;  
    out.matrix_[0] = x_scale;  
    out.matrix_[5] = y_scale;  
    out.matrix_[10] = -((far_plane + near_plane)  
        / frustum_length);  
    out.matrix_[11] = -1;  
    out.matrix_[14] = -((2 * near_plane * far_plane)  
        / frustum_length);  
  
    return out;  
}
```