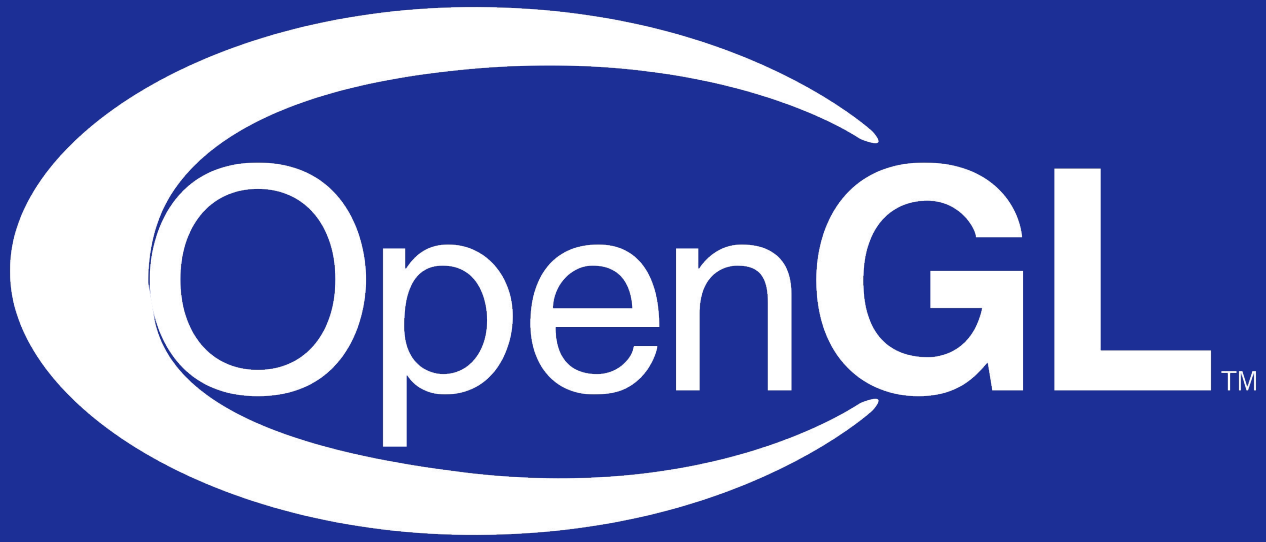
Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

OpenGL : Prymitywy, obsługa klawiatury



mgr inż. Michał Chwesiuk
mgr inż. Tomasz Sergej
inż. Patryk Piotrowski

Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Prymitywy

Implementacja brył

Stos macierzy

Wejście klawiatury

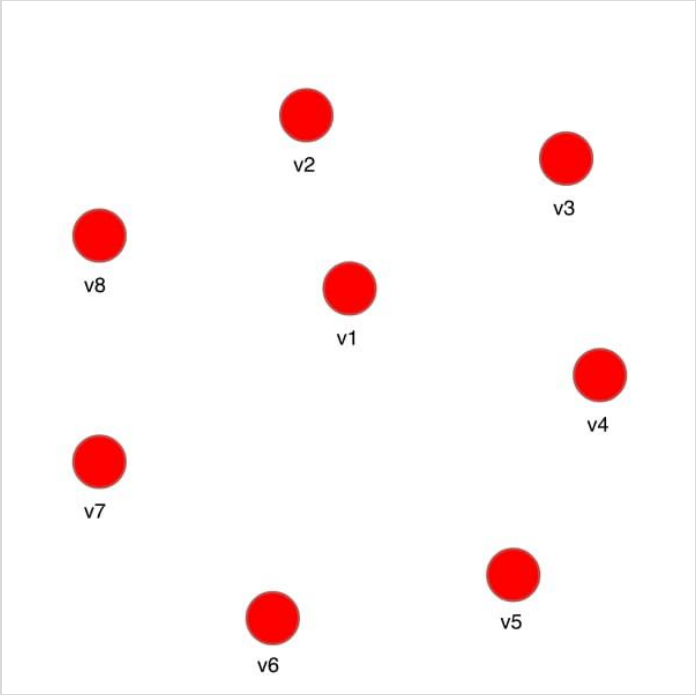
Zadanie

Prymitywy w OpenGL

Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

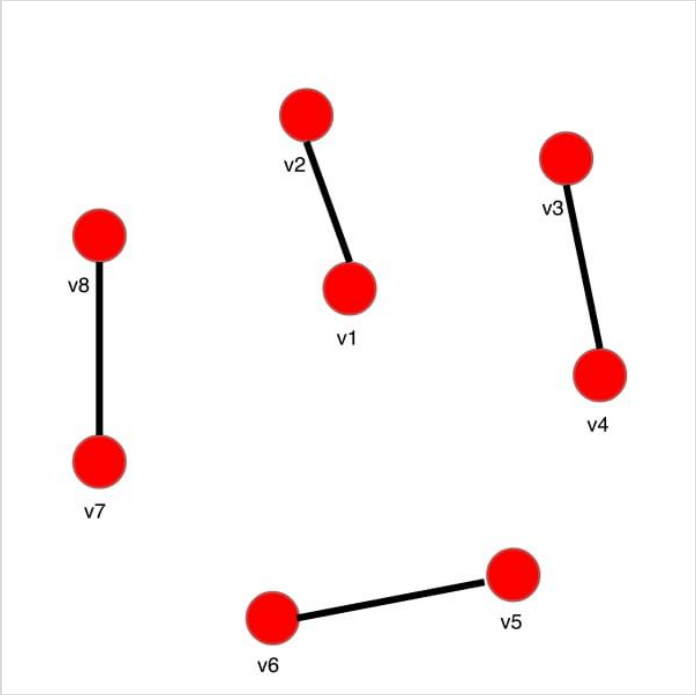
Prymityw - Points (GL_POINTS)



Wydział
Informatyki

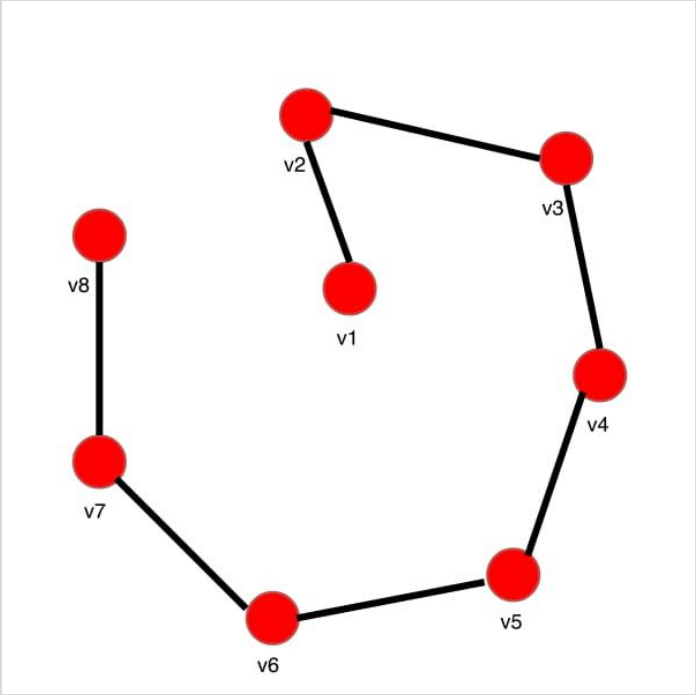
Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Prymityw - Lines (GL_LINES)



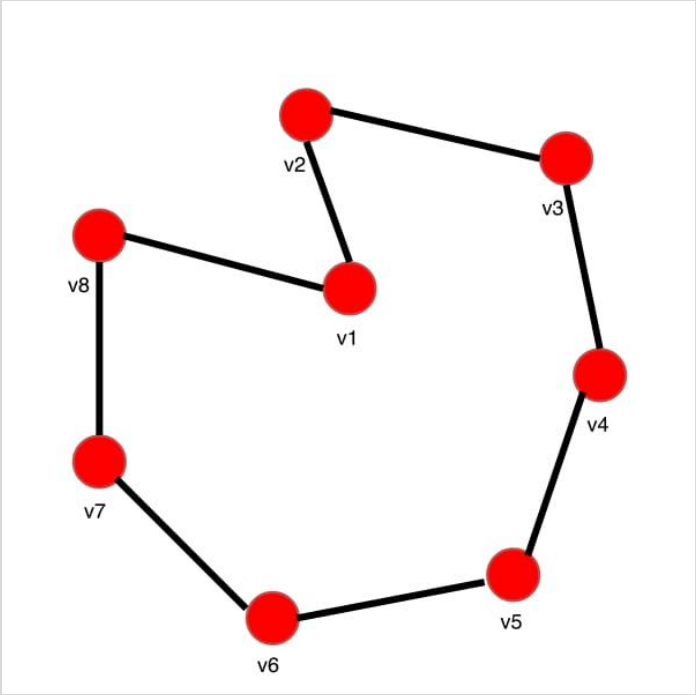
Wydział
Informatyki

Prymityw - Lines Strip (GL_LINE_STRIP)



Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

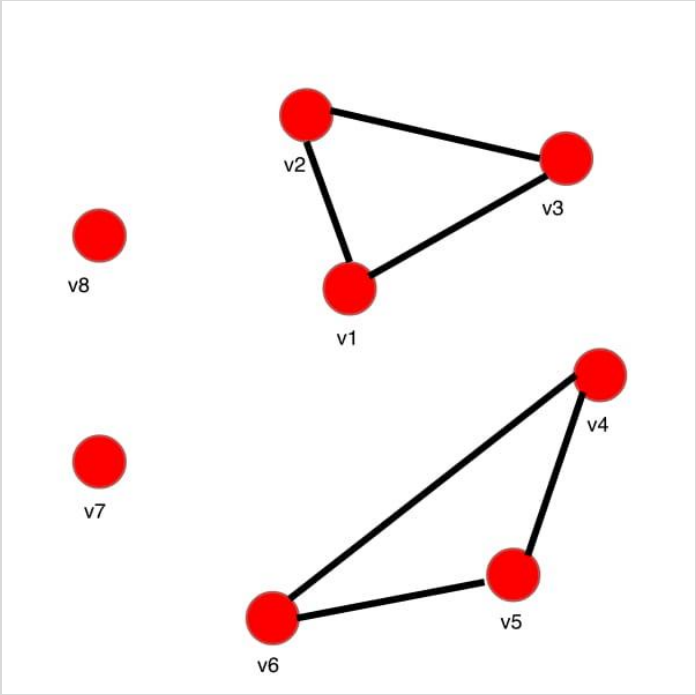
Prymityw - Line Loop (GL_LINE_LOOP)



Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

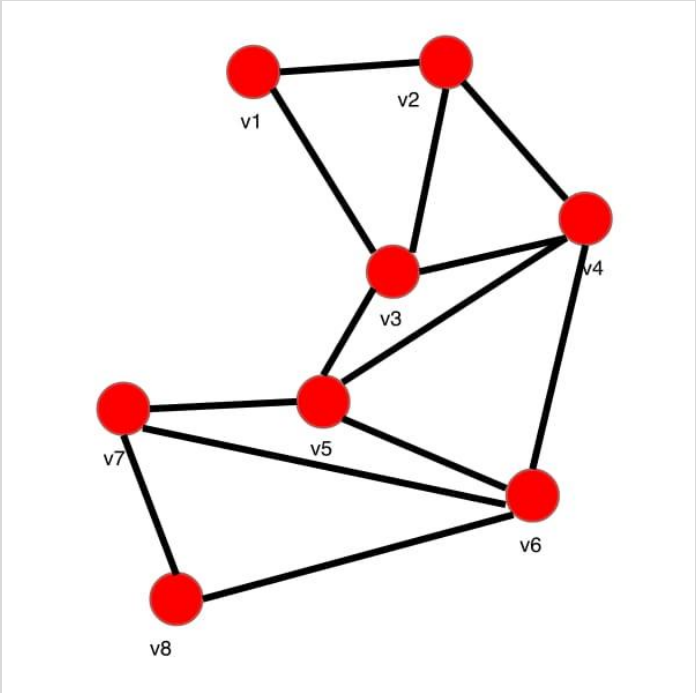
Prymityw - Triangles (GL_TRIANGLES)



Wydział
Informatyki

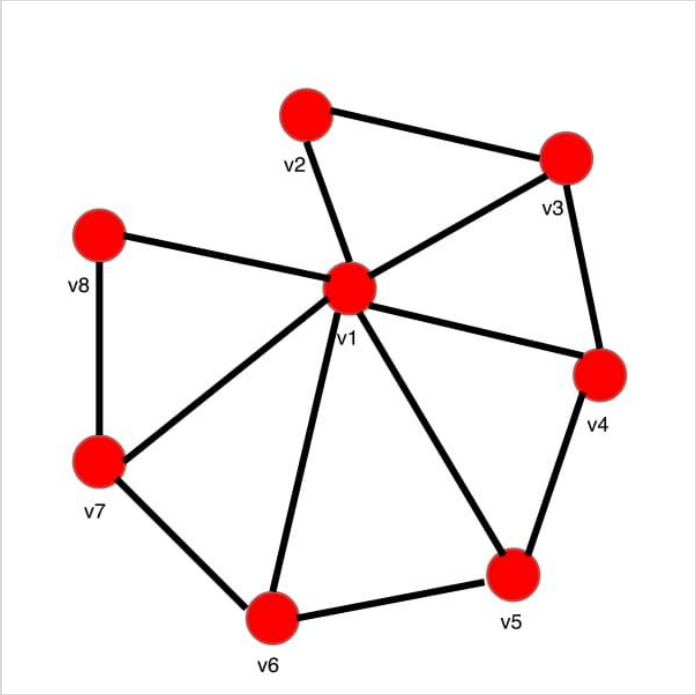
Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Prymityw - Triangle Strip (GL_TRIANGLE_STRIP)



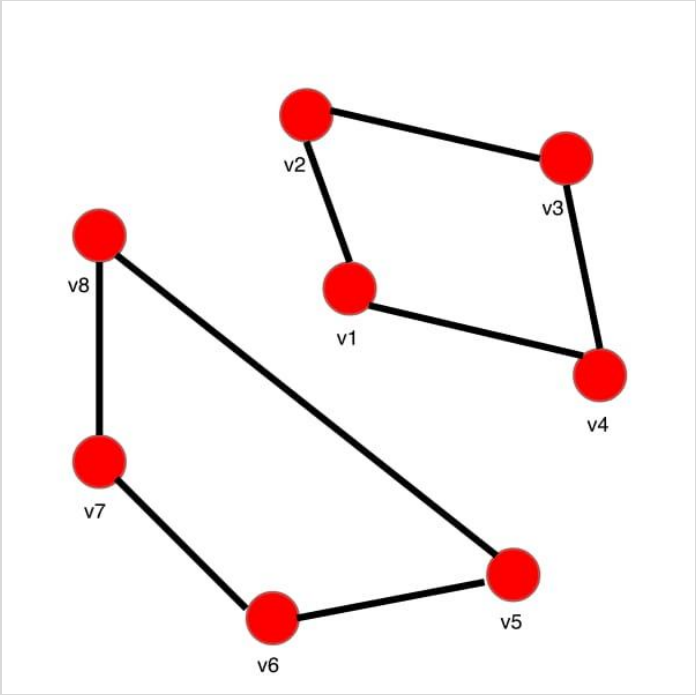
Wydział
Informatyki

Prymityw - Triangle Fan (GL_TRIANGLE_FAN)



Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

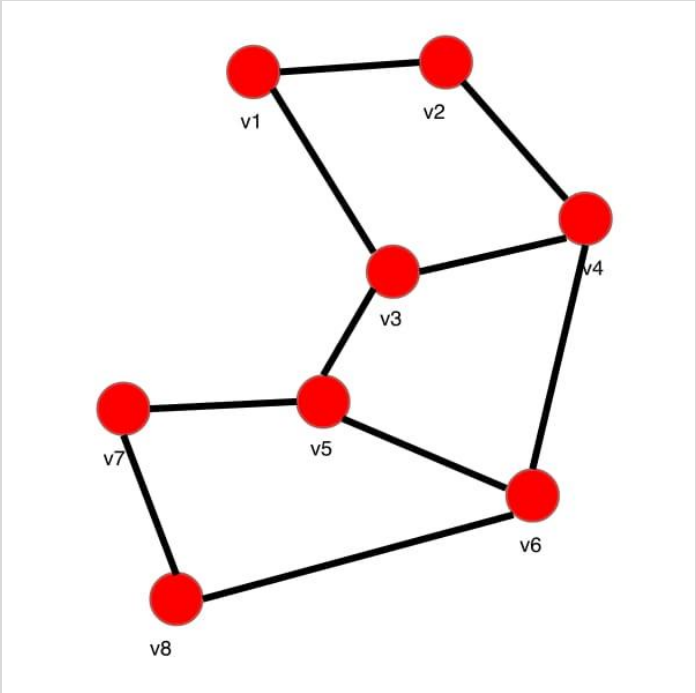
Prymityw - Quads (GL_QUADS)



Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

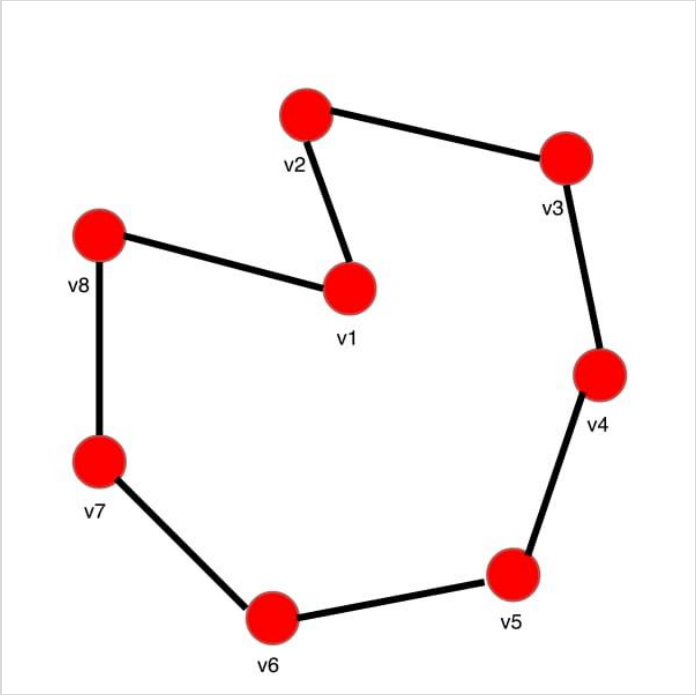
Prymityw - Quad Strip (GL_QUAD_STRIP)



Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Prymityw - Polygon (GL_POLYGON)



Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Implementacja brył

Wydział
Informatyki

Implementacja kuli

Instrukcja opisuje w jaki sposób rozszerzyć kod programu aby umożliwić rendering trójwymiarowej sfery.

1. Otwórz plik ***cobject.h*** i do klasy CObject dodaj deklaracje funkcji void generateSphere(float r, int N). Następnie zaimplementuj tę funkcję w pliku ***cobject.cpp*** (kod na kolejnym slajdzie).

2. W pliku ***glwidget.h*** w klasie GLWidget zadeklaruj zmienne :
   a. CObject m_obj_sphere;
   b. QOpenGLVertexArrayObject m_vao_sphere;
   c. QOpenGLBuffer m_objVbo_sphere;

3. W pliku ***glwidget.cpp*** wewnątrz funkcji *initializeGL*() zainicjuj sferę bazującą na VBO i VAO.
   a. m_obj_sphere.generateSphere(0.5, 24);
   b. m_vao_sphere.create();
   c. QOpenGLVertexArrayObject::Binder vaoBinderSphere(&m_vao_sphere);
   d. m_objVbo_sphere.create();
   e. m_objVbo_sphere.bind();
   f. m_objVbo_sphere.allocate(m_obj_sphere.constData(), m_obj_sphere.count() * sizeof(GLfloat));
   g. setupVertexAttribs();

4. Aby wyrenderować sferę, użyć VAO należące do sfery wewnątrz funkcji *paintGL*() w pliku ***glwidget.cpp***
   a. QOpenGLVertexArrayObject::Binder vaoBinderSphere(&m_vao_sphere);
   b. m_world_stack.push(m_world);
   c. //Transformacje macierzy modelu (translate, rotate, scale)
   d. setTransforms();
   e. vaoBinderSphere.rebind();
   f. glDrawArrays(m_obj_sphere.primitive(), 0, m_obj_sphere.vertexCount());
   g. m_world = m_world_stack.pop();

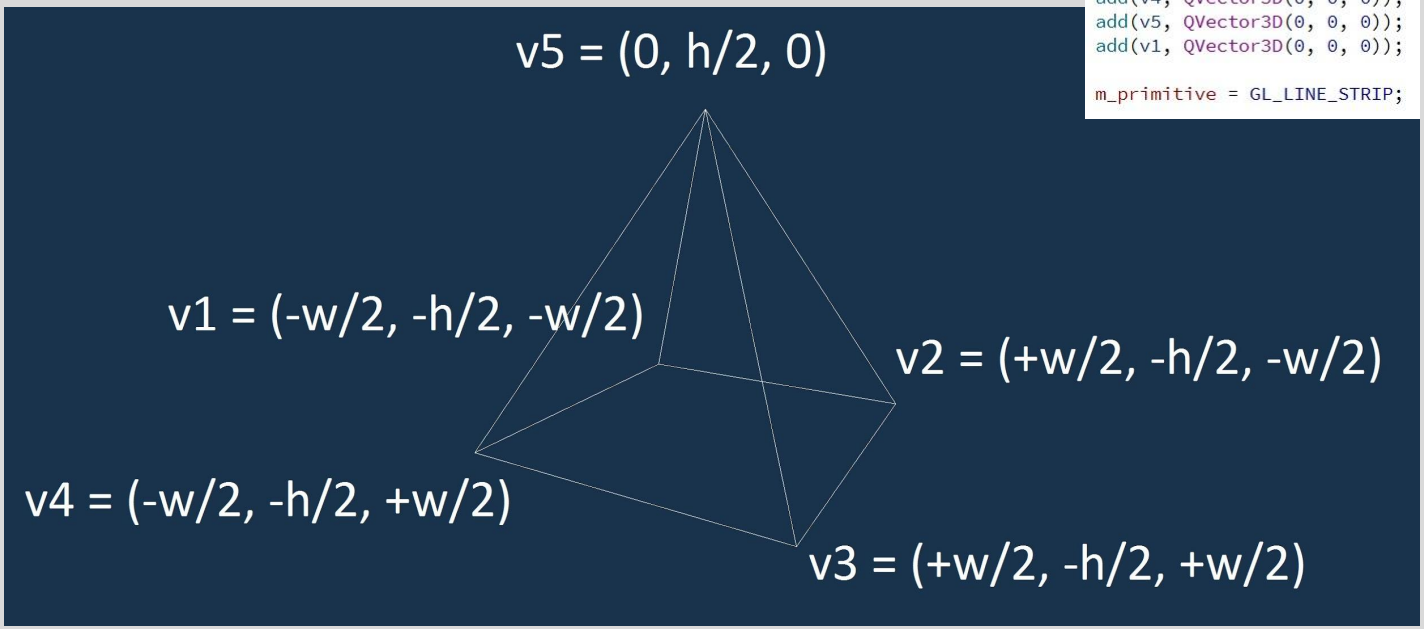
Implementacja kuli


```
void CObject::generateSphere(float r, int N)
{
  for (int j = 0; j <= N; ++j) {
    for (int i = 0; i <= N; ++i) {

      float θ1= π * j / N;
      float θ2 = π * (j - 1) / N;
      float φ = π * 2 * i / N;

      float nx1 = sin(θ1) * cos(φ);
      float nx2 = sin(θ2) * cos(φ);
      float nz1 = sin(θ1) * sin(φ);
      float nz2 = sin(θ2) * sin(φ);
      float ny1 = cos(θ1);
      float ny2 = cos(θ2);

      float x1 = r * nx1;
      float x2 = r * nx2;
      float z1 = r * nz1;
      float z2 = r * nz2;
      float y1 = r * ny1;
      float y2 = r * ny2;

      add(QVector3D(x1, y1, z1), QVector3D(nx1, ny1, nz1));
      add(QVector3D(x2, y2, z2), QVector3D(nx2, ny2, nz2));
    }
  }
  m_primitive = GL_QUAD_STRIP;
}
```

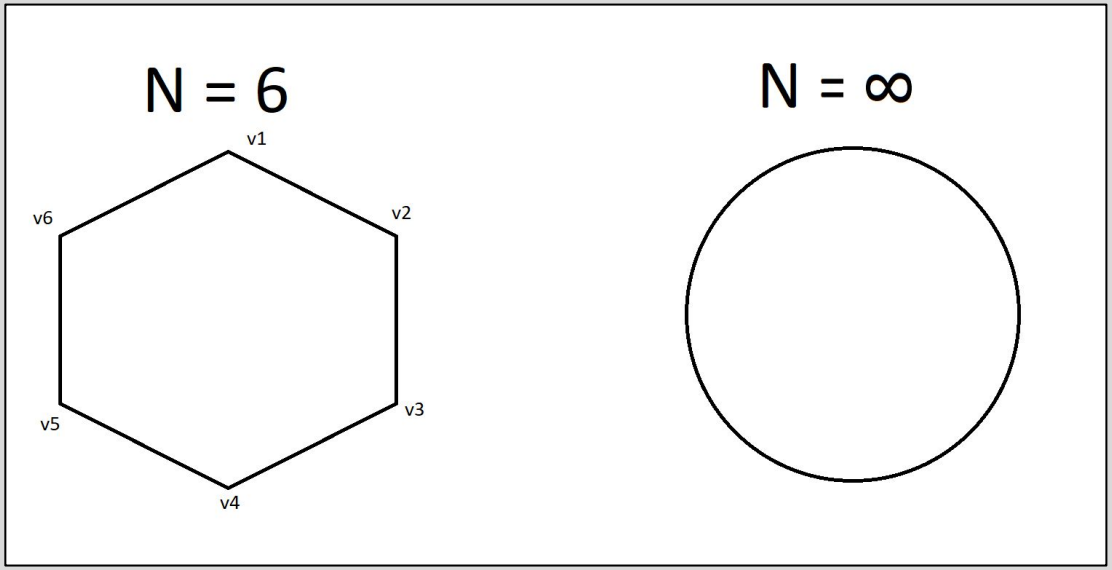
Implementacja ostrosłupa

- Zaimplementować funkcje generatePiramid(float w, float h) w klasie *cobject.h*
- Schemat dodawania nowej bryły do projektu podobny jak w przypadku sfery.

```
add(v1, QVector3D(0, 0, 0));
add(v2, QVector3D(0, 0, 0));
add(v3, QVector3D(0, 0, 0));

add(v3, QVector3D(0, 0, 0));
add(v4, QVector3D(0, 0, 0));
add(v1, QVector3D(0, 0, 0));

add(v1, QVector3D(0, 0, 0));
add(v5, QVector3D(0, 0, 0));
add(v2, QVector3D(0, 0, 0));

add(v2, QVector3D(0, 0, 0));
add(v5, QVector3D(0, 0, 0));
add(v3, QVector3D(0, 0, 0));

add(v3, QVector3D(0, 0, 0));
add(v5, QVector3D(0, 0, 0));
add(v4, QVector3D(0, 0, 0));

add(v4, QVector3D(0, 0, 0));
add(v5, QVector3D(0, 0, 0));
add(v1, QVector3D(0, 0, 0));

m_primitive = GL_LINE_STRIP;
```



Implementacja walca (podstawa)



v1 = QVector3D(radius * cos(0 * 2 * π / 6), H, radius * sin(0 * 2 * π / 6))

v2 = QVector3D(radius * cos(1 * 2 * π / 6), H, radius * sin(1 * 2 * π / 6))

v3 = QVector3D(radius * cos(2 * 2 * π / 6), H, radius * sin(2 * 2 * π / 6))

v4 = QVector3D(radius * cos(3 * 2 * π / 6), H, radius * sin(3 * 2 * π / 6))

v5 = QVector3D(radius * cos(4 * 2 * π / 6), H, radius * sin(4 * 2 * π / 6))

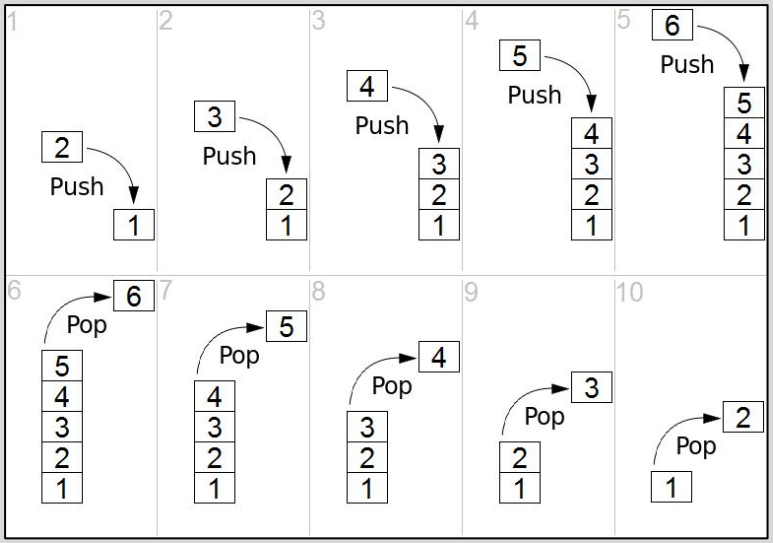
v6 = QVector3D(radius * cos(5 * 2 * π / 6), H, radius * sin(5 * 2 * π / 6))

Stos macierzy

Stos macierzy

Implementacja stosu macierzy:

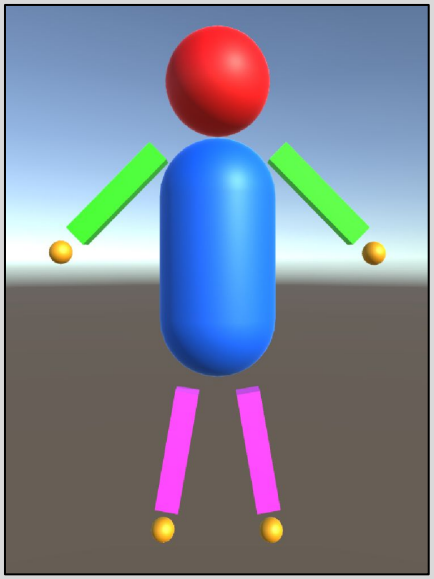
- **#include <qstack.h>** - biblioteka zawierająca implementację stosu.
- **QStack<QMatrix4x4> m_world_stack** - deklaracja stosu macierzy 4x4.
- **m_world_stack.push(m_world)** - dodanie macierzy na wierzch stosu.
- **m_world = m_world_stack.pop()** - pobranie macierzy ze stosu.



Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Hierarchia stosu macierzy


```
[CharacterTransform]
PushMatrix();
        [Head]
        PushMatrix();
                SetTransforms(Translate->Rotate->Scale);
                DrawHead();
        PopMatrix();
        [Body]
        PushMatrix();
                SetTransforms(Translate->Rotate->Scale);
                DrawBody();
        PopMatrix();
        [LeftArmTransform]
        PushMatrix();
                SetTransforms(Translate);
                [LeftArm]
                PushMatrix();
                        SetTransforms(Translate->Rotate->Scale);
                        DrawLeftArm();
                PopMatrix();
                [LeftPalm]
                PushMatrix();
                        SetTransforms(Translate->Rotate->Scale);
                        DrawLeftPalm();
                PopMatrix();
        PopMatrix();
        [RightArmTransform]
        ....
        [LeftLegTransform]
        PushMatrix();
                SetTransforms(Translate);
                [LeftLeg]
                PushMatrix();
                        SetTransforms(Translate->Rotate->Scale);
                        DrawLeftLeg();
                PopMatrix();
                [LeftFoot]
                PushMatrix();
                        SetTransforms(Translate->Rotate->Scale);
                        DrawLeftFoot();
                PopMatrix();
        [RightLegTransform]
        ....
PopMatrix();
```



Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Wejście klawiatury

Wydział
Informatyki

Wejście klawiatury

Instrukcja opisuje w jaki sposób rozszerzyć kod programu aby umożliwić poruszanie się obiektem w scenie za pomocą klawiatury.

1. Otwórz plik ***glwidget..h*** i do klasy CObject dodaj dwie zmienne.
   a. QVector3D playerPosition; // Wektor opisujący pozycję gracza.
   b. char keyState[256]; // Tablica stanów klawiszy.

2. W pliku ***glwidget.cpp*** przed rysowaniem obiektów (w funkcji *paintGL*()), ale wewnątrz bloku "Push/Pop Matrix" przesuń obiekt o vector playerPosition (użyj funkcji *translate* na macierzy modelu/świata*)*.

3. W pliku ***glwidget.cpp*** w funkcji *keyPressEvent*(QKeyEvent *e) ustaw keyState[e->key()] na true
   a. Wymagane sprawdzenie czy e->key() jest w przedziale [0 ; 255].

4. W pliku ***glwidget.cpp*** w funkcji *keyReleaseEvent*(QKeyEvent *e) ustaw keyState[e->key()] na false
   a. Wymagane sprawdzenie czy e->key() jest w przedziale [0 ; 255].

5. Aby poruszać obiektem do przodu po wciśnięciu przycisku W należy dodać poniższy kod na końcu funkcji *paintGL*()
   a. if(keyState[Qt::*Key_W*]) playerPosition.setZ(playerPosition.z() - 0.02);

P.S. Szybkość aktualizacji świata w tym przykładzie jest zależna od szybkości renderingu. **Jest to błędne podejście!** Aktualizacja świata powinna być niezależna od szybkości renderingu.

Zadanie

Wersja podstawowa - 0.75pkt

- Stworzyć obiekt przedstawiający sferę, i narysować “macierz sfer” na scenie.

```
for(int i = - 5 ; i <= 5 ; i++)
    for(int j = -5 ; j <= 5 ; j++)
    {
        m_world_stack.push(m_world);
        m_world.translate(i * 0.5f, -0.5f, j * 0.5f);
        m_world.scale(0.1f, 0.1f, 0.1f);
        setTransforms();
        vaoBinderSphere.rebind();
        m_program->setUniformValue(m_modelColorLoc, QVector3D(i*0.1+0.5, j*0.1+0.5, 0.0));
        glDrawArrays(m_obj_sphere.primitive(), 0, m_obj_sphere.vertexCount());
        m_world = m_world_stack.pop();
    }
```

- Zaimplementować ostrosłup.
- Zaimplementować poruszanie się postaci używając przycisków W,S,A,D.

Wersja rozszerzona - 1.00pkt

- Wersja podstawowa.
- Implementacja walca i stożka.