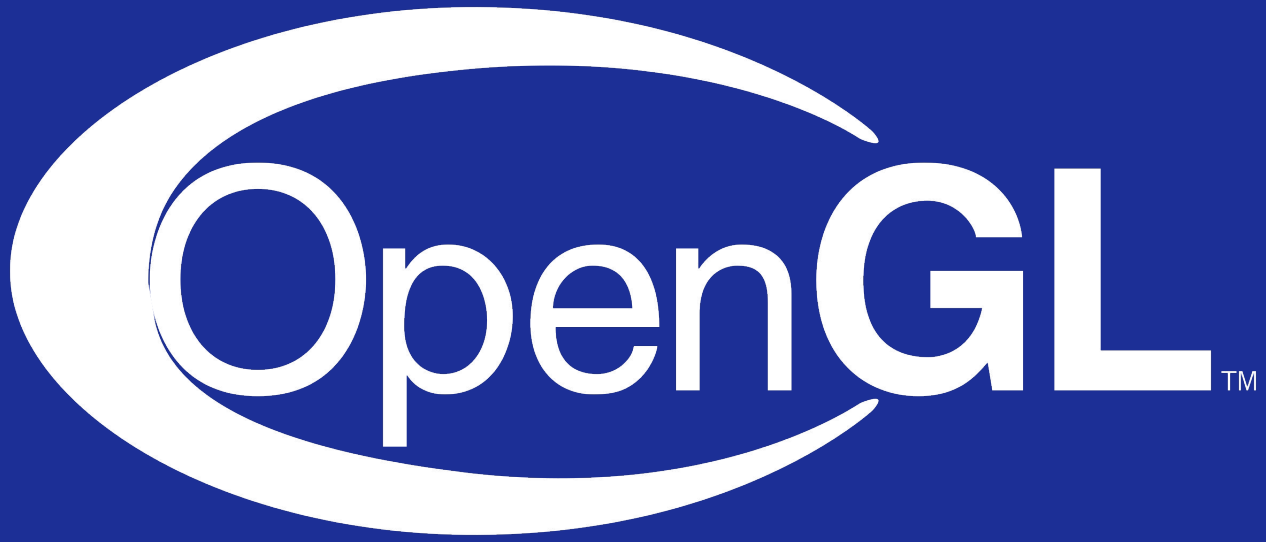
Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Gry Komputerowe - laboratorium 2

Kamera FPP / TPP



Wydział
Informatyki

mgr inż. Michał Chwesiuk

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Kamera FPP vs TPP

Wydział
Informatyki

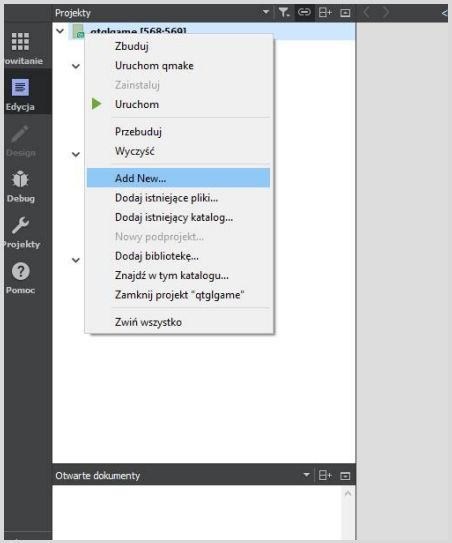
Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

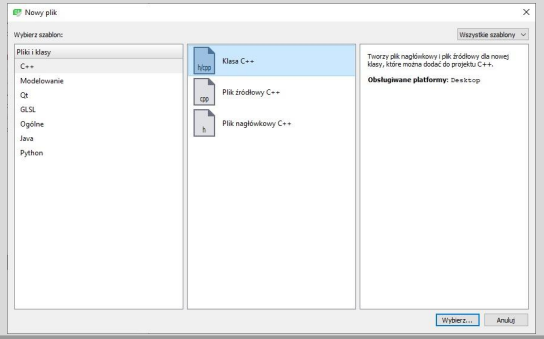
Klasa Player

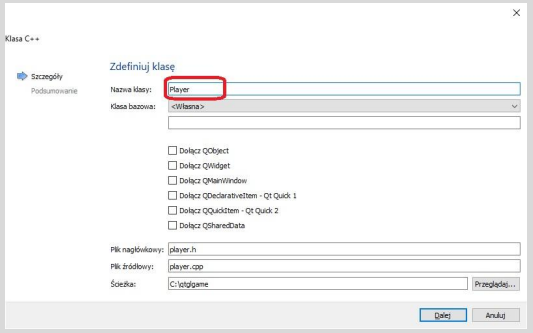
- Stwórz nową klasę Player
- Do stworzonej klasy **Player** w pliku player.h dodaj trzy pola (trzeba dodać #include <QVector3D>).

  QVector3D position;
  QVector3D direction;
  float speed;

- W ciele konstruktora ustaw te pola na domyślne wartości.
  - position = QVector3D(0, 0, 0);
  - direction = QVector3D(0, 0, -1);
  - speed = 0.01f;
- Do klasy **GLWidget** dodaj pole typu player (należy także w pliku glwidget.h dodać #include<player.h>)
  - *Player m_player;*



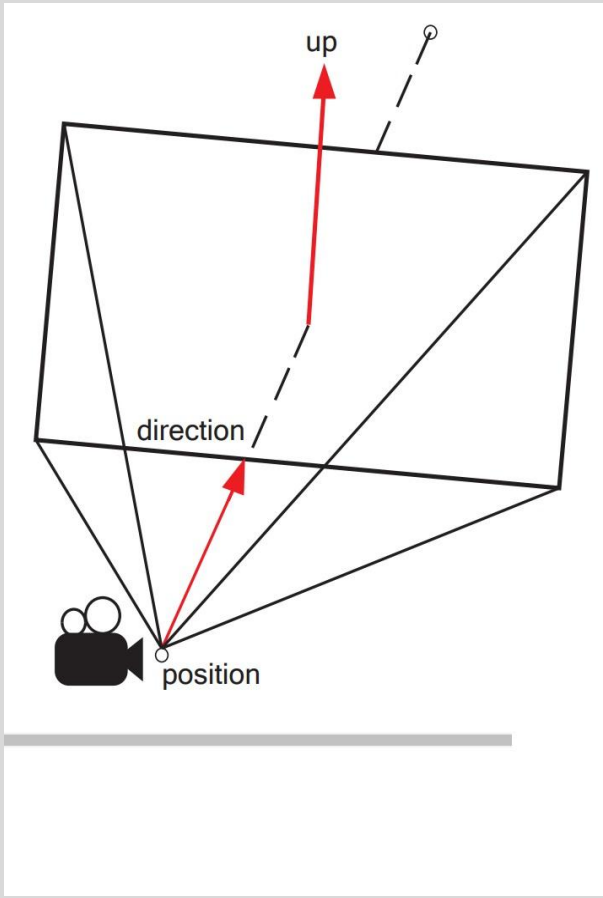




Wydział
Informatyki

Kamera

- Funkcja *lookAt()* pozwala na ustawienie macierzy widoku w taki sposób, że będziemy obserwować scenę z danego punktu w danym kierunku. Funkcja ta przyjmuje trzy parametry :
  - Eye - pozycja kamery
  - Target - punkt w który patrzy kamera
  - Up - wektor góry
- Parametry *eye* i *target* ustawiamy w zależności od rodzaju kamery.
- First Person Perspective
  - Eye = position
  - Target = position + direction
- Third Person Perspective
  - Eye = Position - Direction
  - Target = Position
- Wektor góry ustawiamy na wektor, który symbolizuje górę na naszej scenie.
  - Domyślna wartość : [0, 1, 0]



Kamera

- W funkcji **GLWidget::paintGL()** wyszukujemy poprzednie ustawienie kamery

```
m_camera.setToIdentity();
m_camera.translate(0, 0, -m_camDistance);
m_world.setToIdentity();
m_world.rotate(m_camXRot, 1, 0, 0);
m_world.rotate(m_camYRot, 0, 1, 0);
m_world.rotate(m_camZRot, 0, 0, 1);
```

- i zamieniamy na ustawienie kamery FPP

```
m_camera.lookAt(
                m_player.position,
                m_player.position + m_player.direction,
                QVector3D(0, 1, 0) );
```

- bądź kamery TPP

```
m_camera.lookAt(
                m_player.position - m_camDistance * m_player.direction,
                m_player.position,
                QVector3D(0, 1, 0) );
```

- Przed *m_camera.lookAt()* dodajemy

```
m_camera.setToIdentity();
m_world.setToIdentity();
```

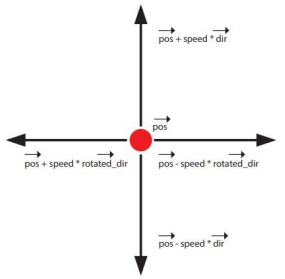
Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Sterowaniem graczem

- Należy zaimplementować poruszaniem się graczem za pomocą klawiatury zależnego od wektora *direction*.
- Przykład poruszania do przodu (wstawić do funkcji *updateGL()*)

```
if(m_keyState[Qt::Key_W])
{
        m_player.position.setX(m_player.position.x() + m_player.direction.x() * m_player.speed);
        m_player.position.setZ(m_player.position.z() + m_player.direction.z() * m_player.speed);
}
```

- W podobny sposób można zaimplementować poruszanie się do tyłu, oraz na boki.
- Do tyłu :
  - *posX = posX - dirX * speed*
  - *posZ = posZ - dirZ * speed*
- W lewo :
  - *posX = posX + dirZ * speed*
  - *posZ = posZ - dirX * speed*
- W prawo :
  - *posX = posX - dirZ * speed*
  - *posZ = posZ + dirX * speed*



- $\vec{pos}$ - pozycja kamery
- $\vec{dir}$ - kierunek patrzenia kamery
- $speed$ - prędkość przemieszczania kamery
- $\vec{rotated\_dir}$ - wektor prostopadły do kierunku patrzenia kamery $\vec{dir}$

Wydział Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Obrót kamery w jednej osii

- Obrót kamery według jednej osii można osiągnąć za pomocą współrzędnych biegunowych.
- Najczęściej podczas opisywania punktu w przestrzeni dwuwymiarowej wykorzystujemy współrzędne kartezjańskie, które opisują punkt za pomocą dwóch zmiennych : X i Y.
- Współrzędne biegunowe polegają na opisaniu tych współrzędnych za pomocą dwóch innych atrybutów :
  - promień wodzący r - odległość punktu P od środka układu współrzędnych O
  - amplituda punktu φ - kąt pomiędzy osią OX, a wektorem OP

Transformacje punktów do układów

- Z układu kartezjańskiego do układu biegunowego

$$r = \sqrt{X^2 + Y^2}$$

$$\phi = arctan(\frac{Y}{X})$$

- Z układu biegunowego do układu kartezjańskiego

$$X = r * cos(\phi)$$

$$Y = r * sin(\phi)$$

Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Obrót kamery w jednej osii

- W funkcji *updateGL()* dodać poniższe linie kodu :

```
if(m_keyState[Qt::Key_Q])
   {
      float phi = atan2(m_player.direction.z(), m_player.direction.x());
      phi = phi - 0.05;
      m_player.direction.setX(cos(phi));
      m_player.direction.setZ(sin(phi));
   }
```

- Należy usunąć z funkcji *keyPressEvent()* :

```
else if (e->key() == Qt::Key_Q)
      exit(0);
```

- Można dodać obrót w drugą stronę za pomocą klawisza *E*.

Wydział
Informatyki

Obrót kamery w dwóch osiach

- Obrót kamery według dwóch osii można osiągnąć za pomocą współrzędnych sferycznych.
- Do przedstawienia tych współrzędnych bardzo pomocna jest wikipedia.
- Przekształcenie do układu sferycznego

```
r = sqrt(X^2, Y^2, Z^2) // w naszym przypadku zawsze równe 1, nie trzeba liczyć
phi = atan2(Z, X)
theta = acos(Y / r)
```

- Przekształcenie z układu sferycznego

```
X = r * sin(theta) * cos(phi)
Y = r * cos(theta)
Z = r * sin(theta) * sin(phi)
```

- Aby obracać kamerą za pomocą myszy, można wewnątrz funkcji *mouseMoveEvent()* użyć zadeklarowanych tam zmiennych *dx* oraz *dy*.

```
// obliczenie phi i theta
phi = phi + dx * 0.01;
theta = theta + dy * 0.01;
// ustawienie m_player.direction
```

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Obrót kamery w dwóch osiach

- Przydatne funkcje do użycia :
  - *setMouseTracking(true);*
    - w konstruktorze *GLWidget*.
    - Powoduje ciągłe sprawdzanie przesunięć kursora.
  - QCursor::setPos(mapToGlobal(QPoint(width()/2, height()/2)));
    - w *updateGL()*
    - Ustawia co klatkę pozycję kursora na środek.
    - Wymagana zmiana przeliczenia *dx* i *dy* w *mouseMoveEvent()*

```
int dx = event->x() - width()/2;
int dy = event->y() - height()/2;
```

  - QCursor c = cursor();
    c.setShape(Qt::CursorShape::BlankCursor);
    setCursor(c);
    - w konstruktorze *GLWidget*.
    - Ukryje kursor.
- Warto też sprawdzić czy obrót kamery w osii Y nie wychodzi po za “bezpieczny obszar” (w funkcji *mouseMoveEvent()* po edycji *theta* względem *dy*).

```
if(theta < 0.01) theta = 0.01;
if(theta > 3.14) theta = 3.14;
```

Wydział
Informatyki

Zachodniopomorski
Uniwersytet Technologiczny
w Szczecinie

Kamera TPP a model gracza

- Można dodać poruszanie się robotem za pomocą zaimplementowanej klasy *player*.
- W miejscu gdzie był kod przesunięcia robota (wewnątrz jego bloku push/pop

  m_world.translate(m_robotPosition);

  Wstawić translację o wektor *m_player.position*.

  m_world.translate(m_player.position);
- Aby robot był obrócony w kierunku wektora *direction* gracza, należy w tym samym miejscu po translacji wstawić obrót o kąt *phi* według osii Y wyliczony za pomocą konwersji do współrzędnych sferycznych, której używaliśmy do obrotu kamery.
- Ten kąt musi być skonwertowany w radianów na stopnie, oraz przemnożony przez -1, należy także dodać 90.

  float phi = atan2(m_player.direction.z(), m_player.direction.x());
  *m_world.rotate(-phi * 180.0f / M_PI + 90, 0, 1, 0);*
- Przez wprowadzenie dynamicznego obrotu obiektów na scenie, należy zmienić przeliczenie wektorów normalnych w programach cieniujących (shaderach).
- W pliku **shader.vs** zamień linijkę `fragNormal = normal;` na
  - `fragNormal = (modelMatrix * vec4(normal, 0)).xyz;`

Wydział
Informatyki