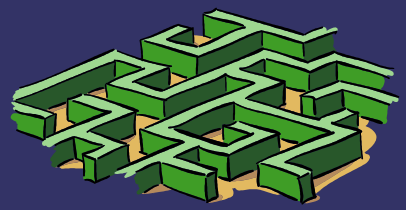
Введение

О чём речь

Тема доклада: мой личный опыт создания универсальной камеры на C++ для 3D редактора. Результатом стал компактный класс, реализующий FPS, Target и Orbit камеры без использования кватернионов и без хранения матриц.

Содержание

- Задача
- Теория вращения мира
- Реализация вращения камеры
- Немного математики
- Завершение



Задача

Цели и задачи

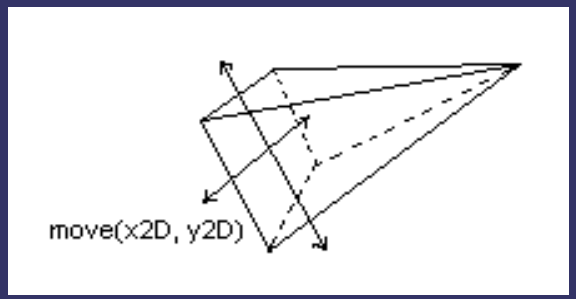
Релизовать камеру для 3D редактора, которая несёт в себе функции всех трёх видов камер: FPS, Target и Orbit. А так же, для 3D редактора необходимо:

- сохранять и восстанавливать из файла позицию камеры, позицию цели и значения углов наклона камеры, которые вводил пользователь через графический интерфейс;

- перемещать камеру вперёд и назад;

- перемещаться влево-вправо (стрейф) и вверх-вниз для создания возможности “таскать” по экрану рабочую трёхмерную область, не меняя направление камеры;

- так же будет уместно поместить сюда рассчёт перспективной матрицы.

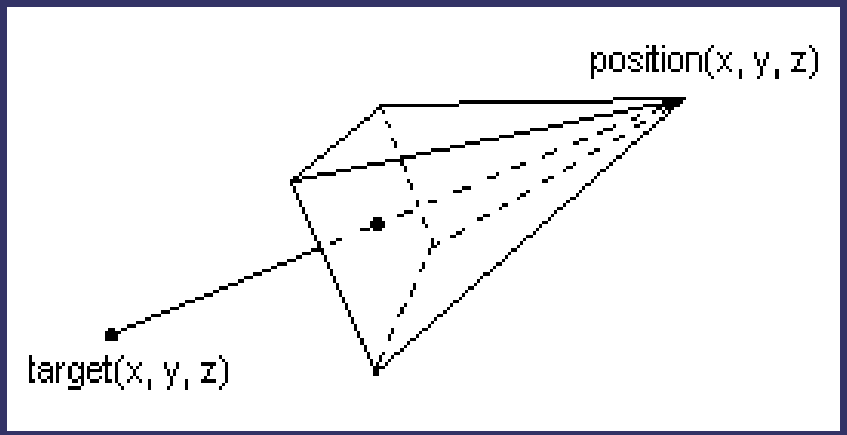


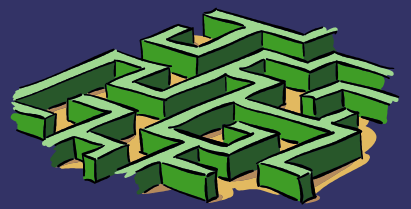
Задача

Target camera

Смотрит на цель, заданную координатами X, Y, Z

Праметры:
float x0, y0, z0 – позиция камеры
float x1, y1, z1 – позиция цели





Задача

FPS camera

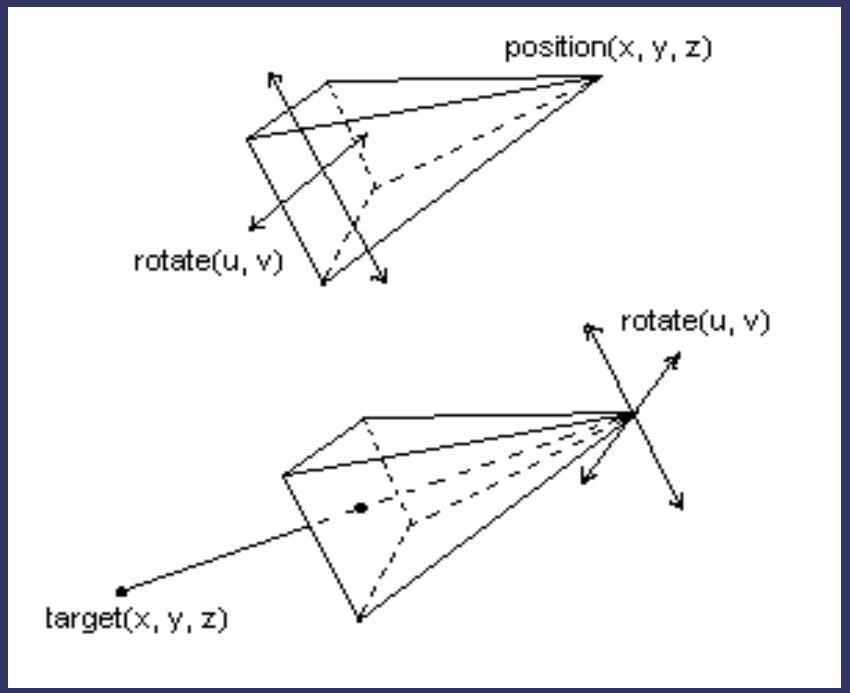
Камера от первого лица.

Параметры:
float x, y, z – позиция камеры
float u, v – углы поворота

Orbit camera

Вращение во круг заданной цели.

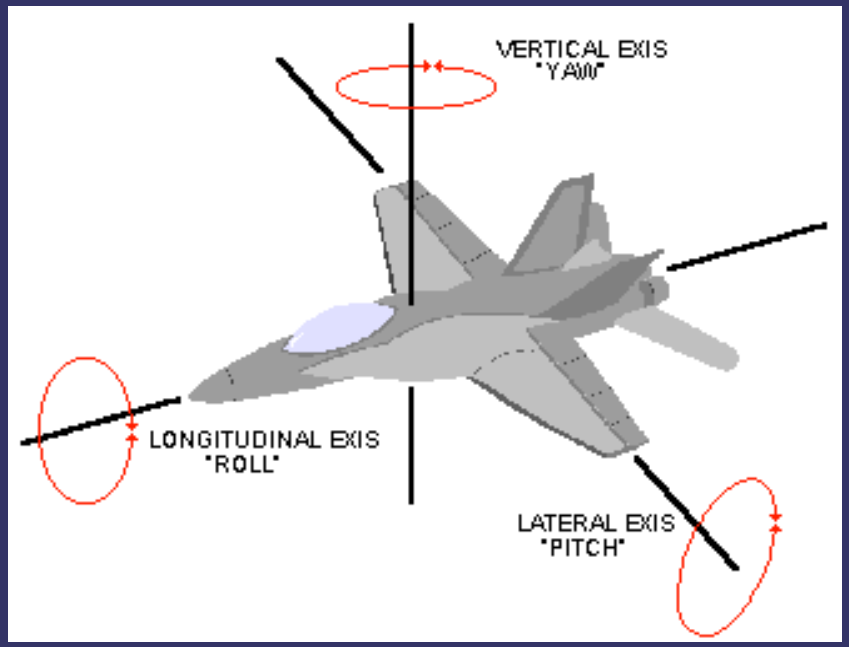
Параметры:
float x, y, z – позиция цели
float u, v – углы поворота во круг цели



Задача

Углы поворота

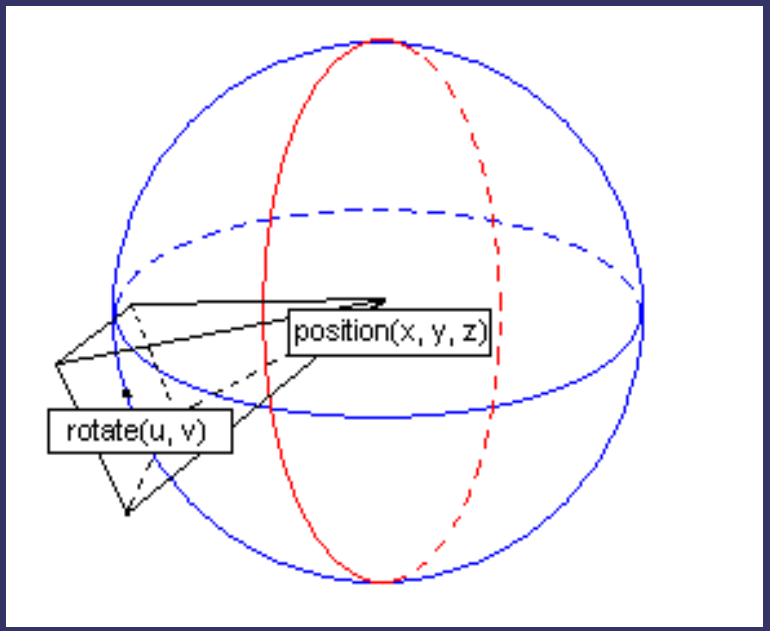
Углы поворота U, V можно представить как двухмерные координаты точки на сфере (долгота, широта; зенит, азимут; тангаж, рысканье; yaw, pitch).



Задача

Углы поворота

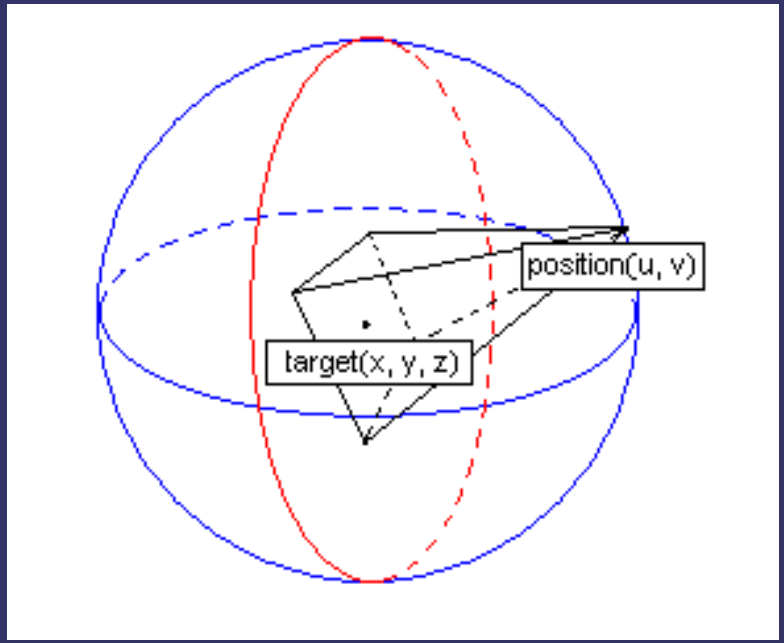
Оси вращения на примере FPS камеры отображены синим цветом. По оси красного цвета камера должна быть зафиксирована всё время.



Задача

Углы поворота

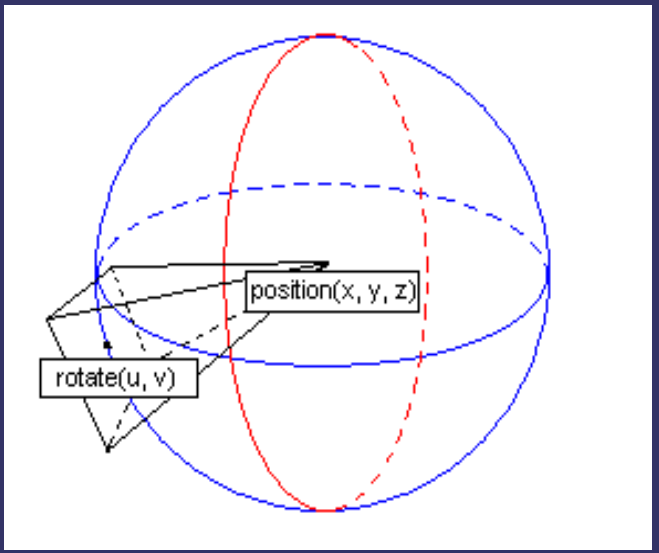
Для Orbit камеры необходимо рассчитывать положение камеры (x, y, z) согласно углам вращения во круг цели.

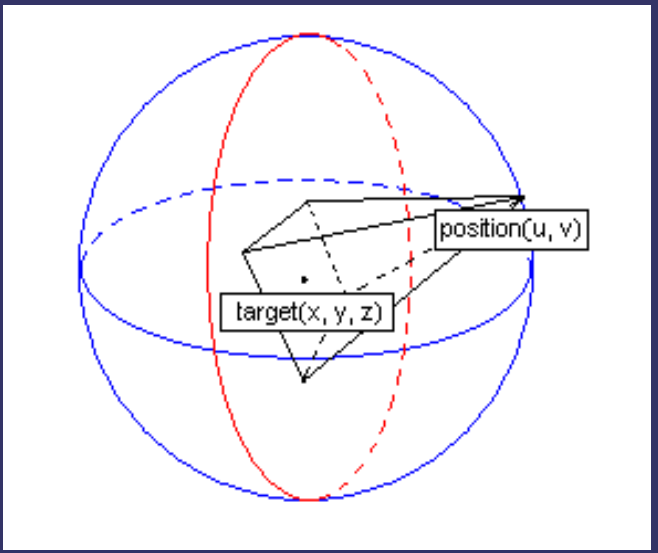


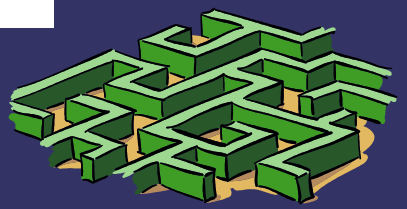
Задача

Обобщение FPS и Orbit

Замечаем тот факт, что если в orbit камере точка цели будет совпадать с положением камеры, то эффект будет такой же, как и при вращении FPS камеры.







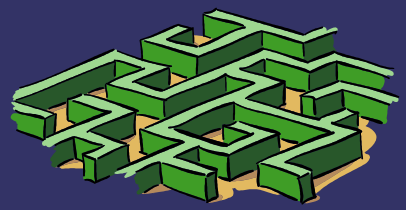
Теория вращения мира

Способы вращения пространства

Необходимо иметь математические средства для рассчёта и хранения данных о вращении. Для этого существуют всего 4 варианта:

- Углы Эйлера
- Матрица поворота
- Кватернион
- Как нибудь ещё..

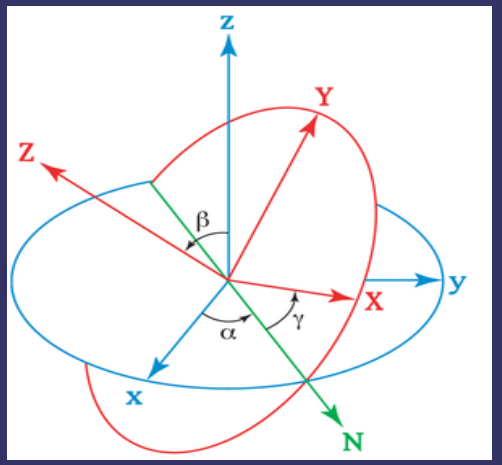
Рассмотрим каждый из них в отдельности.



Теория вращения мира

Углы Эйлера

Углы Эйлера определяют три поворота, которые позволяют привести любое положение системы к текущему. Такие повороты некоммутативны и конечное положение системы зависит от порядка, в котором совершаются повороты.



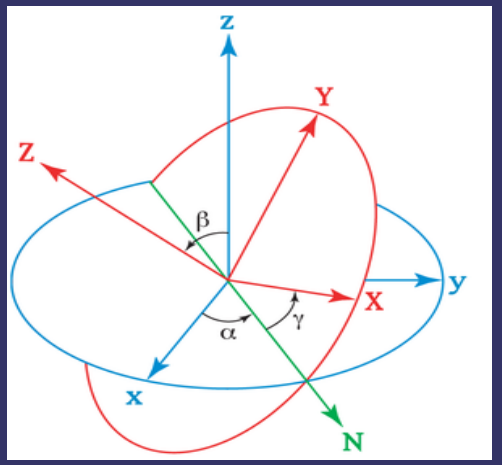
Любопытно, что если повернуть кубик на 45° по всем трём осям, то диагональ кубика НЕ будет параллельна оси OZ.

Каждый последующий поворот выполняется не относительно глобальных осей, а относительно локальных осей обьекта, относительно уже выполненного поворота.

Теория вращения мира

Углы Эйлера

Углы Эйлера определяют три поворота системы, которые позволяют привести любое положение системы к текущему. Такие повороты некоммутативны и конечное положение системы зависит от порядка, в котором совершаются повороты.



Любопытно, что если повернуть кубик на 45 градусов по всем трём осям, то диагональ кубика НЕ будет параллельна оси OZ.

Каждый последующий поворот выполняется не относительно глобальных осей, а относительно локальных осей обьекта, относительно уже выполненного поворота.

Теория вращения мира

Матрица поворота

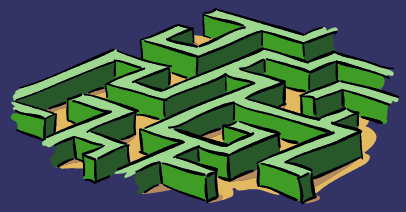
Умножив вектор на матрицу поворота, получим новый вектор, повернутый на заданные углы относительно координатных осей.

Любое вращение в трехмерном пространстве может быть представлено произведением соответствующих трех матриц поворота.

$$M_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}, \quad M_y(\alpha) = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix}, \quad M_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

В результат произведения - матрица поворота в 3D пространестве.

$$M(\alpha,\beta,\gamma) = \begin{pmatrix} \cos\alpha\cos\gamma - \sin\alpha\cos\beta\sin\gamma & -\cos\alpha\sin\gamma - \sin\alpha\cos\beta\cos\gamma & \sin\alpha\sin\beta \\ \sin\alpha\cos\gamma + \cos\alpha\cos\beta\sin\gamma & -\sin\alpha\sin\gamma + \cos\alpha\cos\beta\cos\gamma & -\cos\alpha\sin\beta \\ \sin\beta\sin\gamma & \sin\beta\cos\gamma & \cos\beta \end{pmatrix}$$



Теория вращения мира

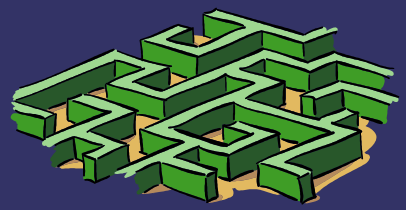
Матрица поворота

В трёхмерной графике используется матрица поворота, выраженная через угол поворота θ и единичный вектор оси вращения (x, y, z).

$$M(\hat{\mathbf{v}},\theta) = \begin{pmatrix} \cos\theta + (1-\cos\theta)x^2 & (1-\cos\theta)xy - (\sin\theta)z & (1-\cos\theta)xz + (\sin\theta)y \\ (1-\cos\theta)yx + (\sin\theta)z & \cos\theta + (1-\cos\theta)y^2 & (1-\cos\theta)yz - (\sin\theta)x \\ (1-\cos\theta)zx - (\sin\theta)y & (1-\cos\theta)zy + (\sin\theta)x & \cos\theta + (1-\cos\theta)z^2 \end{pmatrix}$$

Например, в OpenGL есть функции glRotate*(**angl, x, y, z**) для установки угла поворота и функции glMultMatrix*(...) для установки заранее подготовленной матрицы поворота. Других функций для вращения нет.

Проблема заключается в том, что при использовании обратных тригонометрических функций (arctg, arcctg) для вычисления углов поворота по матрице, мы получаем угол в диапазоне от 0 до 180. Данные о симмитричных углах от 180 до 360 теряются.



Теория вращения мира

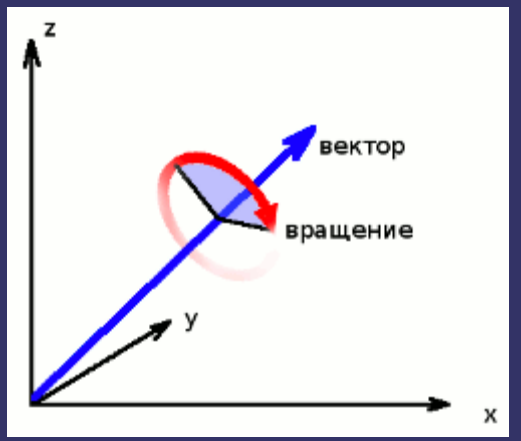
Кватернион

Кватернион можно построить из вектора (x, y, z) задающего ось вращения объекта, и угла поворота w. Это однозначно определяет поворот объекта.

Кватернионы позволяют:

- выполнять поворот вокруг оси, независимо от совершённого вращения по другим осям;

- избежать неопределённости при вычислении углов поворота из имеющегося кватерниона.



Теория вращения мира

Как нибудь ещё..

Но наиболее интересным представляется способ “как нибудь ещё”, который и является темой этого доклада, и будет описан далее.

Реализация вращения камеры

Переменные класса камеры

Поскольку мы пишем 3D редактор, позицию камеры мы храним не в виде кватерниона, не в виде матрицы, а в виде набора конечных переменных, которые задал пользователь через графический интерфейс. Это нужно для восстановления точных значений в полях ввода.

Данные, которыми мы будем оперировать при рассчёте матрицы вида:

```
float m_targetX, m_targetY, m_targetZ;
float m_posX, m_posY, m_posZ;
float m_rotX, m_rotY;
```

Реализация вращения камеры

Построение выходной матрицы

Результирующая матрица строится по углам поворота и положению камеры. Это означает, что:

- при установке положения цели нужно рассчитывать углы поворота;
- при установке углов наклона как то корректировать положение цели;
- при вращении вокруг цели рассчитывать новое положение камеры.

```
void CameraUniversal::p_updateView()
{
    matrixDL viewMatrix;
    viewMatrix.set_angle( TO_RADIAN( m_rotX ), 0.0f, 0.0f );
    viewMatrix.set_angle_append( 0.0f, 0.0f, TO_RADIAN( m_rotY ) );
    viewMatrix.set_translate( vec3DL( m_posX, m_posY, m_posZ ) );
}
```

Реализация вращения камеры

Перечень функций для вращения:

```
void setRotationOrbit( float rotXOrbit, float rotYOrbit );
```

m_rotX = user, m_rotY = user;
m_posX = formula, m_posY = formula, m_posZ = formula;
m_targetX = nope, m_targetY = nope, m_targetZ = nope;

```
void setRotationXY( float rotX, float rotY );
```

m_rotX = user, m_rotY = user;
m_posX = nope, m_posY = nope, m_posZ = nope;
m_targetX = formula, m_targetY = formula, m_targetZ = formula;

```
void setTarget( float posXTarget, float posYTarget, float posZTarget );
```

m_rotX = formula, m_rotY = formula;
m_posX = nope, m_posY = nope, m_posZ = nope;
m_targetX = user, m_targetY = user, m_targetZ = user;

Реализация вращения камеры

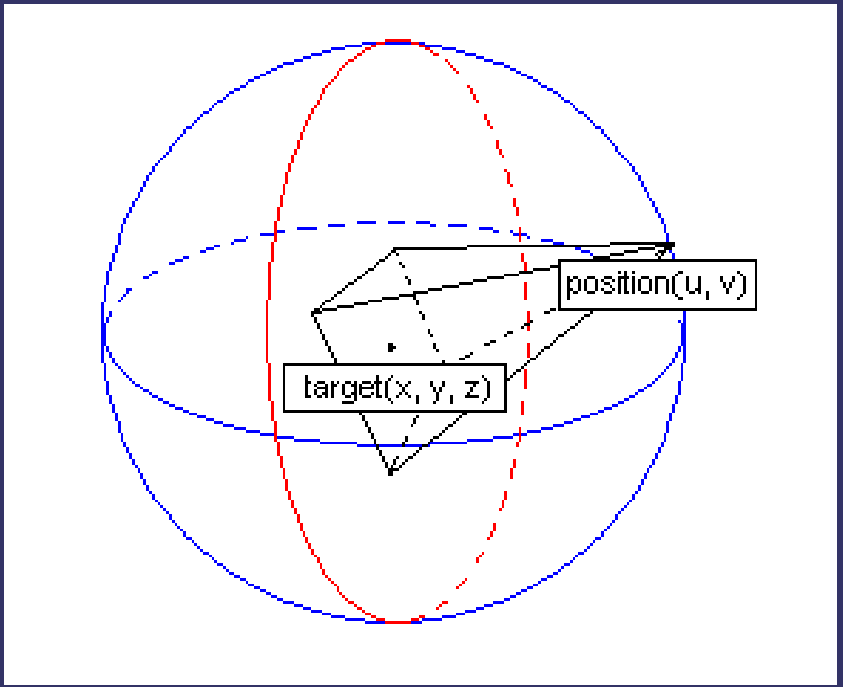
Вращение FPS

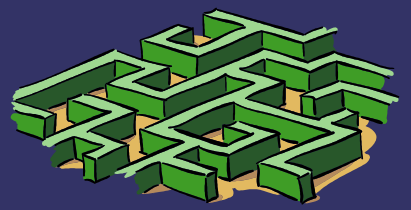
```
void CameraUniversal::setRotationXY( float rotX, float rotY )
{
    m_rotX = rotX; m_rotY = rotY;

    double R = len( m_posX, m_posY, m_posZ, \
                    m_targetX, m_targetY, m_targetZ );

    double f = TO_RADIAN( m_rotY );
    double t = TO_RADIAN( m_rotX );

    m_targetX = m_posX - sin( t ) * sin( f ) *  R;
    m_targetY = m_posY - sin( t ) * cos( f ) * -R;
    m_targetZ = m_posZ - cos( t ) * R;
}
```

Реализация вращения камеры

Вращение Orbit

Напоминаю, что зная 2D “координаты точки” на сфере, где должна быть камера, по формуле сферы можно рассчитать 3D координаты камеры.

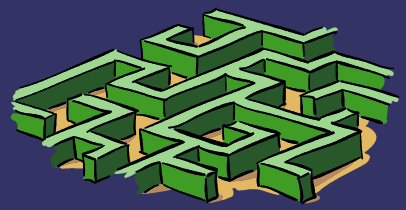




Реализация вращения камеры

Вращение Orbit

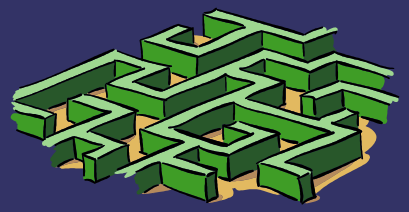
```
void CameraUniversal::setRotationOrbit( float rotXOrbit, float rotYOrbit )
{
    m_rotX = rotXOrbit; m_rotY = rotYOrbit;

    double R = len( m_posX, m_posY, m_posZ, \
                    m_targetX, m_targetY, m_targetZ );

    double f = TO_RADIAN( rotY );
    double t = TO_RADIAN( rotX );

    m_posX = sin( t ) * sin( f ) *  R   + m_targetX;
    m_posY = sin( t ) * cos( f ) * -R   + m_targetY;
    m_posZ = cos( t ) * R               + m_targetZ;
}
```



Реализация вращения камеры

А теперь самое интересное!!

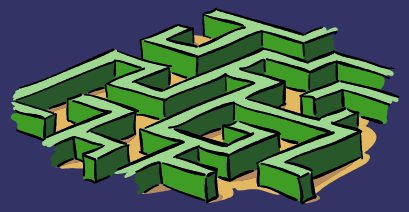
^_^



Реализация вращения камеры

Но сперва немного математики..

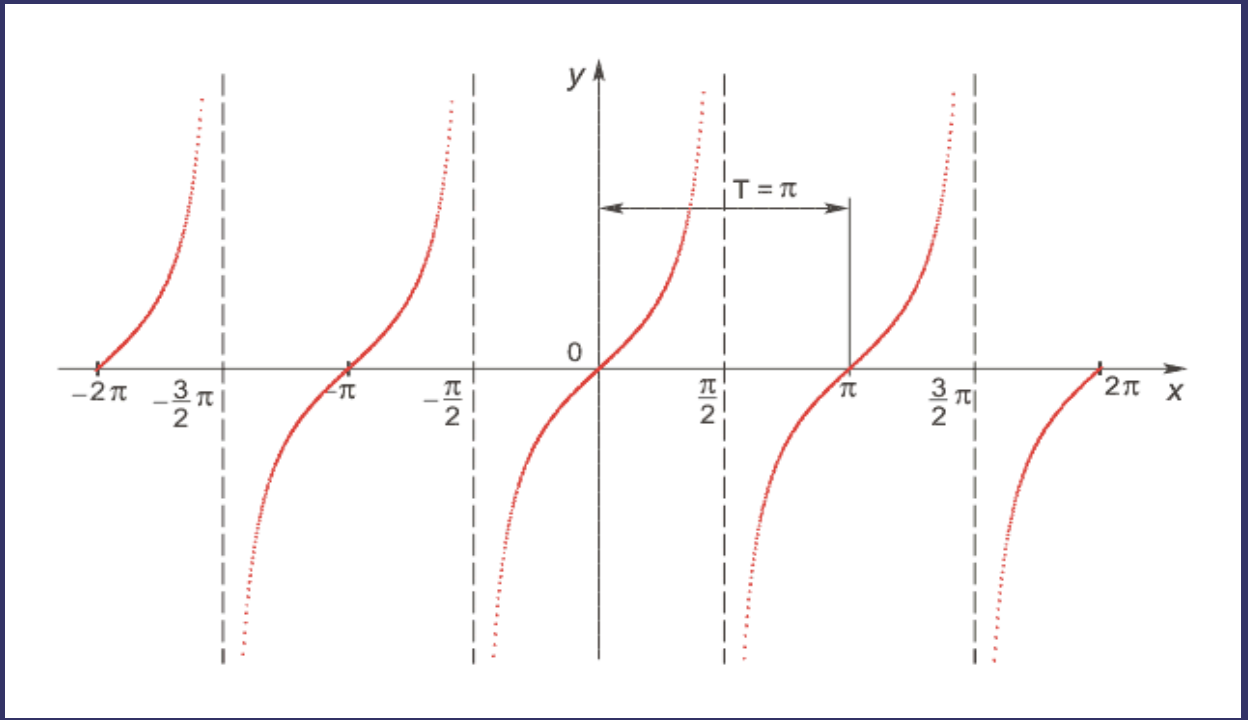
--_--



Немного математики

Тангенс угла

Тангенс – это отношение синуса и косинуса угла (**tan(t) = sin(t) / cos(t)**)
Заметим, что функция тангенса определена при значениях -90° < t < 90°

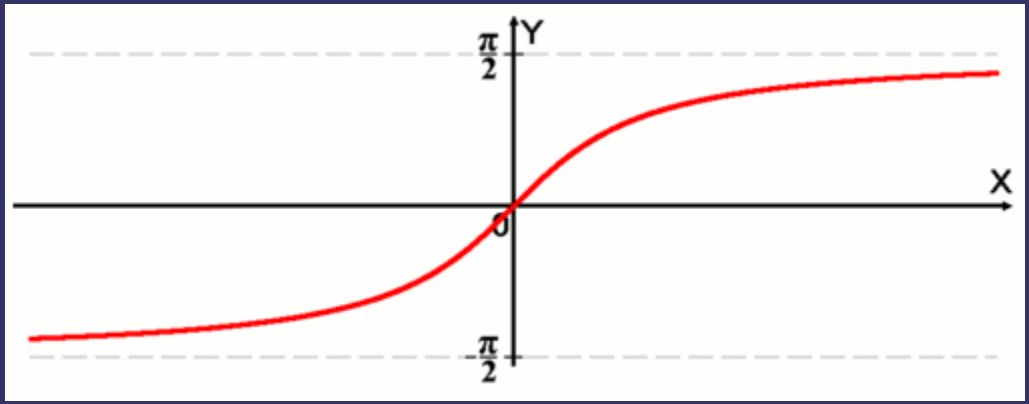


Немного математики

Арктангенс угла

Функция y = arctg(x) непрерывна и ограничена на всей своей числовой прямой; является строго возрастающей.

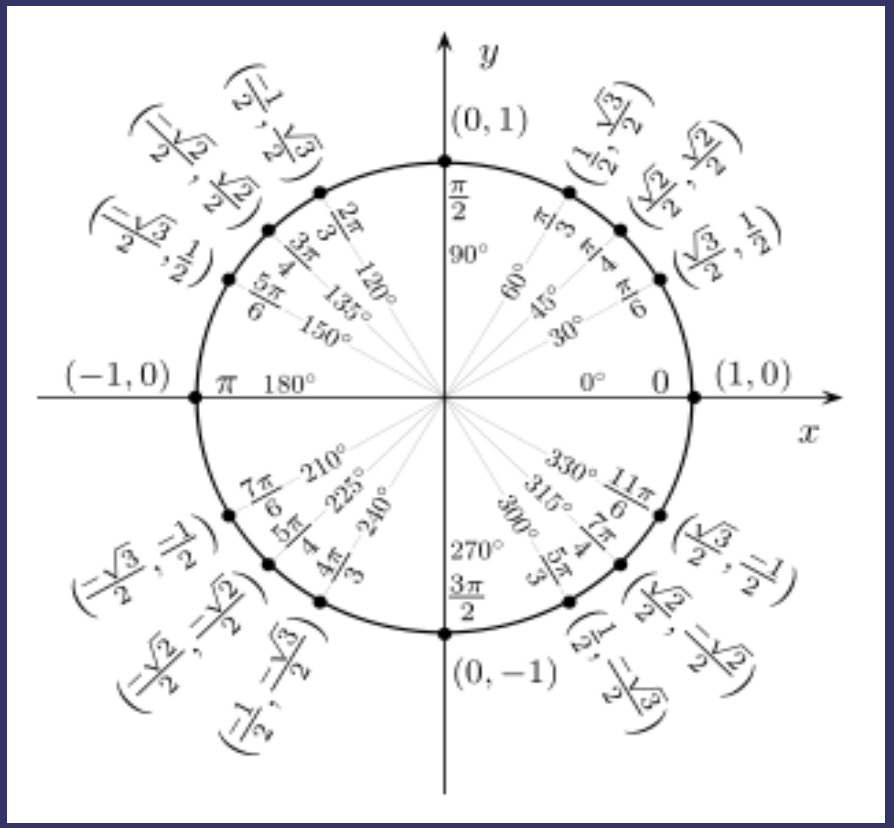
Функция y = tan(x) на всей своей области определения является кусочно-монотонной, и, значит, обратным соответствием функцией y=arctg(x) НЕ является.

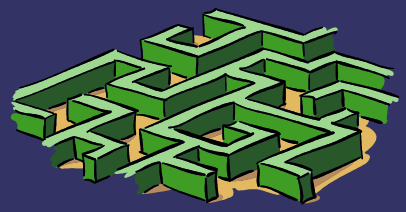


Немного математики

Арктангенс угла

При любых значениях X, функция arctg(X) будет возвращать значение угла в диапазоне от -90° до 90°, что создаёт неопределённость.





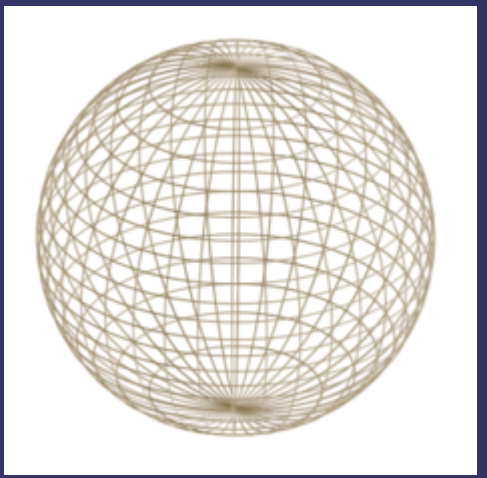
Немного математики

Сферическая система координат

Сферическими координатами называют систему координат для отображения геометрических свойств фигуры в трёх измерениях посредством задания трёх координат (r, θ, φ), где r — расстояние до начала координат, а θ и φ — зенитный и азимутальный угол соответственно.

От сферических к декартовым:

$$\begin{cases} x = r\sin\theta\cos\varphi, \\ y = r\sin\theta\sin\varphi, \\ z = r\cos\theta. \end{cases}$$



От декартовых к сферическим:

$$\begin{cases} r = \sqrt{x^2+y^2+z^2}, \\ \theta = \arccos\left(\dfrac{z}{\sqrt{x^2+y^2+z^2}}\right) = \operatorname{arctg}\left(\dfrac{\sqrt{x^2+y^2}}{z}\right), \\ \varphi = \operatorname{arctg}\left(\dfrac{y}{x}\right). \end{cases}$$

$-90<\varphi<90$

$90<\varphi<270$?

Немного математики

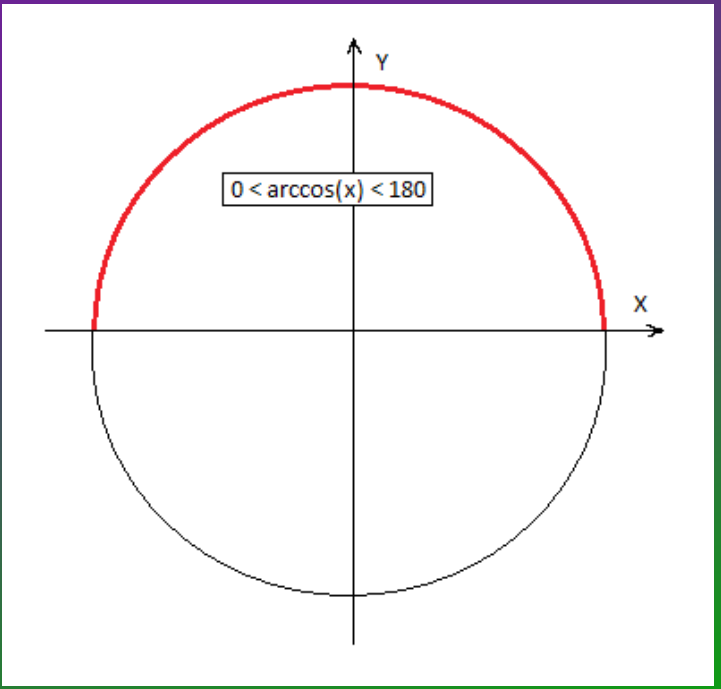
Моё решение

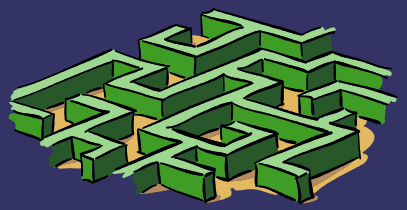
$$X = \sin(t)*\cos(f)$$
$$Y = \sin(t)*\sin(f)$$
$$Z = \cos(t)$$

$$t = \arccos(Z)$$

$$\cos(f) = X / \sin(t)$$
$$f = \arccos(X / \sin(t))$$

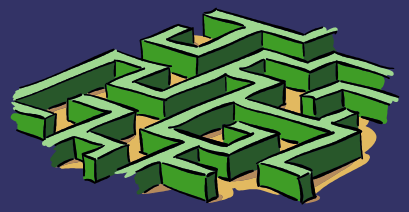
If (Y < 0) f = -f





Реализация вращения камеры

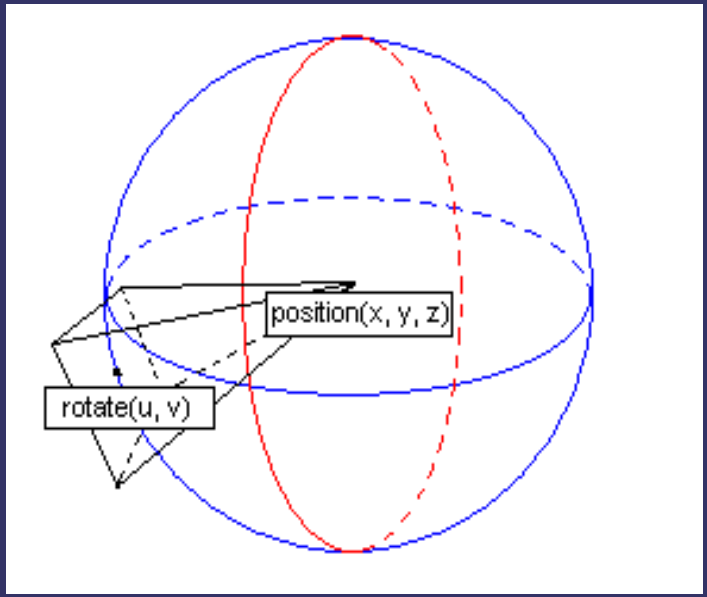
Возвращаемся к реализации камеры на C++



Реализация вращения камеры

Вращение Target

```
void CameraUniversal::setTarget( float posXTarget, float posYTarget, float posZTarget )
{
    vec3DL vecFrom( m_posX,     m_posY,     m_posZ );
    vec3DL vecTo( posXTarget, posYTarget, posZTarget );
    vecFrom -= vecTo; vecFrom.normalize();

    double vecX, vecY, vecZ, vecXpre, vecYpre;
    vecFrom.values( vecX, vecY, vecZ );

    float t = acos( vecZ );

    if( sin( t ) <  GLHDL_EPSILON &&
        sin( t ) > -GLHDL_EPSILON ) return;

    vecXpre =  vecX / sin( t );

    if( fabs( vecXpre ) > 1.0 ) return;

    float f = acos( vecXpre );

    m_rotX = TO_DEGREES( t );
    m_rotY = TO_DEGREES( ( ( vecY < 0 ) ? -f : f ) );

    m_targetX = posXTarget; ...
```



Завершение

Подведём итоги

В результате мы получили вращение камеры во всех трёх видах (FPS, Target, Orbit) без использования кватернионов.

Для хранения параметров камеры в файле используются значения углов поворота, которые ввёл пользователь через графический интерфейс.

При установке углов поворота – перерасчитываюся координаты цели (сохраняя расстояние до цели), при установке цели – перерасчитываются углы поворота.

Полная версия кода с примером использования через Qt в приложении.

--=== Спасибо за внимание! ===--