Putting the Eye Position on an Orbiting Body

mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License

OrbitingEyePosition.pptx
#include <stdio.h>
#include <string>
#define _USE_MATH_DEFINES
#include <cmath>
#define GLM_FORCE_RADIANS
#include "glm/vec2.hpp"
#include "glm/vec3.hpp"
#include "glm/mat4x4.hpp"
#include "glm/gtc/matrix_transform.hpp"
#include "glm/gtc/matrix_inverse.hpp"

#include <GL/gl.h>
#include <GL/glu.h>

enum views
{
    OUTSIDEVIEW, EARTHVIEW, MOONVIEW, CORVALLISVIEW
};

float Time;
enum views WhichView;

Make Time go from 0. to however many seconds you want in your total animation, not 0. to 1.
Timing Setup

At the top of the program:

```cpp
const int MAXIMUM_TIME_SECONDS = 10*60; // I decided to use 10 minutes
const int MAXIMUM_TIME_MILLISECONDS = 1000* MAXIMUM_TIME_SECONDS;
```

In the `Animate( )` function:

```cpp
int ms = glutGet( GLUT_ELAPSED_TIME ); // milliseconds
ms %= MAXIMUM_TIME_MILLISECONDS; // [ 0, MAXIMUM_TIME_MILLISECONDS-1 ]
Time = (float)ms / 1000.f; // seconds
```

// note that Time goes from 0. to however many seconds you asked for, not 0. – 1. !

// force a call to Display( ):
glutSetWindow(MainWindow);
glutPostRedisplay();

In the `InitGraphics( )` function:

```cpp
...  
glutIdleFunc( Animate );
...  
```
Physical Parameter Setup

At the top of the program:

```c
const float EARTH_RADIUS_MILES = 3964.19;
const float EARTH_ORBITAL_RADIUS_MILES = 9290000.;

const float EARTH_ORBIT_TIME_DAYS = 365.3;
const float EARTH_ORBIT_TIME_HOURS = EARTH_ORBIT_TIME_DAYS * 24.;
const float EARTH_ORBIT_TIME_SECONDS = EARTH_ORBIT_TIME_HOURS * 60. * 60.;
const float EARTH_SPIN_TIME_DAYS = 0.9971;
const float EARTH_SPIN_TIME_HOURS = EARTH_SPIN_TIME_DAYS * 24.;
const float EARTH_SPIN_TIME_SECONDS = EARTH_SPIN_TIME_HOURS * 60. * 60.;

const float MOON_RADIUS_MILES = 1079.6;
const float MOON_ORBITAL_RADIUS_MILES = 238900.;

const float MOON_ORBIT_TIME_DAYS = 27.3;
const float MOON_ORBIT_TIME_HOURS = MOON_ORBIT_TIME_DAYS * 24.;
const float MOON_ORBIT_TIME_SECONDS = MOON_ORBIT_TIME_HOURS * 60. * 60.;
const float MOON_SPIN_TIME_DAYS = MOON_ORBIT_TIME_DAYS;
const float MOON_SPIN_TIME_HOURS = MOON_SPIN_TIME_DAYS * 24.;
const float MOON_SPIN_TIME_SECONDS = MOON_SPIN_TIME_HOURS * 60. * 60.;
```

*Warning:* these are the actual numbers for our solar system. You would need to change them to your exaggerated numbers!
Let’s make this fairly straightforward. In model coordinates:

1. Put the eye-**position** at the corner of its Equator and Prime Meridian (xe = EARTH_RADIUS_MILES, ye = 0., ze = 0.)

2. Set the look-at **position** to be on a straight-line east of the eye-position: (xl = EARTH_RADIUS_MILES, yl = 0., zl = -1000.)

3. Set the up-**vector** to be: (xu = 1000., yu = 0., zu = 0.)

This makes our view-vector (from the eye-position to the look-at position) tangent to the Earth’s surface, which is a good way to start.

Now, all we have to do is transform those 3 locations/vector into Solar System Coordinates (I hate to call them “World Coordinates” here...).
Earth Transformations

Steps to transform the Earth-eye-viewing system into Solar System Coordinates:
Using OsuSphere() draw the Earth into a display list at (0.,0.,0.), i.e., the Sun’s center
1. Spin the Earth by EarthSpinAngle about its Y axis
2. Translate the Earth by EARTH_ORBITAL_RADIUS_MILES in its X direction
3. Revolve the Earth by EarthOrbitAngle about the Sun’s Y axis

```cpp
glm::mat4 MakeEarthMatrix()
{
    float earthSpinAngle = Time * EARTH_SPIN_TIME_SECONDS * ONE_FULL_TURN;
    float earthOrbitAngle = Time * EARTH_ORBIT_TIME_SECONDS * ONE_FULL_TURN;
    glm::mat4 identity = glm::mat4( 1. );
    glm::vec3 yaxis = glm::vec3( 0., 1., 0. );

    /* 3. */ glm::mat4 erorbity = glm::rotate( identity, earthOrbitAngle, yaxis );
    /* 2. */ glm::mat4 etransx = glm::translate( identity, glm::vec3( EARTH_ORBITAL_RADIUS_MILES, 0., 0. ) );
    /* 1. */ glm::mat4 erspiny = glm::rotate( identity, earthSpinAngle, yaxis );

    return erorbity * etransx * erspiny; // 3 * 2 * 1
}
```
Your Standard Outside View – the One You’ve Been Using Since Week #0

Put this in the Display() function:

```c
void Display()
{
    ...
    glm::mat4 m;
    glm::vec4 eye = glm::vec4( 0., 0., 0., 1. );
    glm::vec4 look = glm::vec4( 0., 0., 0., 1. );
    glm::vec4 up = glm::vec4( 0., 0., 0., 0. );          // vectors don’t get translations

    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity( );
    switch( WhichView )
    {
        ...
        case OUTSIDEVIEW:   // 1st way to set gluLookAt()
            gluLookAt( 0., 0., 3., 0., 0., 0., 0., 1., 0. );
            glRotatef( (GLfloat)Yrot, 0., 1., 0. );
            glRotatef( (GLfloat)Xrot, 1., 0., 0. );
            if( Scale < MINSCALE )
                Scale = MINSCALE;
            glScalef( Scale, Scale, Scale );
            break;
        ...
    }
    ...
```
Earth Transformations

Put this in the Display( ) function:

```c
void Display( )
{
    . . .
    glm::mat4 m;
    glm::vec4  eye = glm::vec4( 0., 0., 0., 1. );
    glm::vec4 look = glm::vec4( 0., 0., 0., 1. );
    glm::vec4   up = glm::vec4( 0., 0., 0., 0. );        // vectors don’t get translations

    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity( );
    switch( WhichView )
    { . . .
        case EARTHVIEW: // 2nd way to set gluLookAt( )
            m = MakeEarthMatrix( );

            // float eye[4] = { EARTH_RADIUS_MILES, 0., 0., 1.};
            eye.x = EARTH_RADIUS_MILES;
            eye = m * eye;

            // float look[4] = { EARTH_RADIUS_MILES, 0., -1000., 1.};
            look.x = EARTH_RADIUS_MILES;
            look.z = -1000.;
            look = m * look;

            // float up[4] = { 1000., 0., 0., 0. };
            up.x = 1000.;
            up = m * up;

            gluLookAt( eye.x, eye.y, eye.z,   look.x, look.y, look.z,   up.x, up.y, up.z );
            break;
        . . .
    }
}
```
Let’s make this fairly straightforward. In model coordinates:

1. Put the eye-**position** at the corner of its Equator and Prime Meridian
   \((x_e = \text{MOON\_RADIUS\_MILES}, y_e = 0., z_e = 0.)\)

2. Set the look-at **position** to be on a straight-line east of the eye-position:
   \((x_l = \text{MOON\_RADIUS\_MILES}, y_l = 0., z_l = -1000.)\)

3. Set the up-**vector** to be: \((x_u = 1000., y_u = 0., z_u = 0.)\)

This makes our view-vector (from the eye-position to the look-at position) tangent to the Moon’s surface, which is a good way to start.

Now, all we have to do is transform those 3 locations/vector into Solar System Coordinates (I hate to call them “World Coordinates” here...).
Moon Transformations

Steps to transform the Moon-eye-viewing system:
Using OsuSphere() draw the Moon into a display list at (0.,0.,0.), i.e., the Sun’s center

1. Spin the Moon by MoonSpinAngle about its Y axis
2. Translate the Moon by MOON_ORBITAL_RADIUS_MILES in its X direction
3. Revolve the Moon by MoonOrbitAngle about the Earth’s Y axis
4. Translate the Earth by EARTH_ORBITAL_RADIUS_MILES in its X direction
5. Revolve the Earth by EarthOrbitAngle about the Sun’s Y axis

glm::mat4 MakeMoonMatrix()
{
    float moonSpinAngle = Time * MOON_SPIN_TIME_SECONDS * ONE_FULL_TURN;
    float moonOrbitAngle = Time * MOON_ORBIT_TIME_SECONDS * ONE_FULL_TURN;
    float earthOrbitAngle = Time * EARTH_ORBIT_TIME_SECONDS * ONE_FULL_TURN;
    glm::mat4 identity = glm::mat4( 1.);
    glm::vec3 yaxis = glm::vec3( 0., 1., 0.);
    /* 5. */ glm::mat4 erorbity = glm::rotate( identity, earthOrbitAngle, yaxis );
    /* 4. */ glm::mat4 etransx = glm::translate( identity, glm::vec3( EARTH_ORBITAL_RADIUS_MILES, 0., 0. ) );
    /* 3. */ glm::mat4 mrorbity = glm::rotate( identity, moonOrbitAngle, yaxis );
    /* 2. */ glm::mat4 mtransx = glm::translate( identity, glm::vec3( MOON_ORBITAL_RADIUS_MILES, 0., 0. ) );
    /* 1. */ glm::mat4 mrspiny = glm::rotate( identity, moonSpinAngle, yaxis );
    return erorbity * etransx * mrorbity * mtransx * mrspiny;               // 5 * 4 * 3 * 2 * 1  [ M_e/s ] * [ M_m/e ]
}

Note that EarthSpinAngle has no effect on the Moon’s matrix
Moon Transformations

Put this in the Display( ) function:

```c
void Display()
{
    ...  
    glm::mat4 m;
    glm::vec4 eye = glm::vec4( 0., 0., 0., 1. );
    glm::vec4 look = glm::vec4( 0., 0., 0., 1. );
    glm::vec4 up = glm::vec4( 0., 0., 0., 1. );  // vectors don't get translations

    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity( );

    switch (WhichView)
    {
        ... 
        case MOONVIEW:  // 3rd way to set gluLookAt( )
            m = MakeMoonMatrix( );

            // float eye[4] = { MOON_RADIUS_MILES, 0., 0., 1. };
            eye.x = MOON_RADIUS_MILES;
            eye = m * eye;

            look.x = MOON_RADIUS_MILES;
            look.z = -1000.;
            look = m * look;

            // float up[4] = {100 0., 0., 0., 0. };
            up.x = 1000.;
            up = m * up;

            gluLookAt( eye.x, eye.y, eye.z, look.x, look.y, look.z, up.x, up.y, up.z );
            break;
        ...
    }
}
```
Transformations In Action!

Images by Christopher Weiner
What if You Want the Eye at Corvallis (or some other arbitrary location)?

Corvallis sits at Latitude 44.57° N x Longitude -123.27° W

Treating lat-long as spherical coordinates and solve for x, y, and z:

```c
float y = sinf( 44.57° ); // 0.702
float xz = cosf( 44.57° ); // 0.712
float x = xz * cosf( 123.27° ); // -0.391
float z = xz * sinf( 123.27° ); // 0.596
```

Then multiply x, y, and z by EARTH_RADIUS_MILES
Let's assume that we want to look straight east from where we are:

- The eye-position will be (x, y, z)
- By the law of perpendicular-to-a-circle, the eye-to-look vector (i.e., look-eye) = (z, 0., -x)
- So, the look-at position will be eye-position + eye-to-look-vector = (x+z, y, z-x)
- The up vector is (x, y, z)

Note that the dot product of these two vectors is 0., which proves they are perpendicular.

This works for any point on the globe, not just Corvallis
Earth/Corvallis Transformations

Put this in the Display( ) function:

```c
display( )
{
    ...  
    glm::mat4 m;
    glm::vec4 eye = glm::vec4( 0., 0., 0., 1. );
    glm::vec4 look = glm::vec4( 0., 0., 0., 1. );
    glm::vec4 up = glm::vec4( 0., 0., 0., 0. );

    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity( );
    switch( WhichView )
    {
    ...
    case CORVALLISVIEW:  // 4th way to set gluLookAt( )
        m = MakeEarthMatrix( );
        // float eye[4] = { x, y, z, 1. };
        eye.x = x;  eye.y = y;  eye.z = z;
        eye = m * eye;
        // float look[4] = { x+z, y, z-x, 1. };
        look.x = x+z;  look.y = y;  look.z = z-x;
        look = m * look;
        // float up[4] = { x, y, z, 0. };
        up.x = x;  up.y = y;  up.z = z;
        up = m * up;

        gluLookAt( eye.x, eye.y, eye.z, look.x, look.y, look.z, up.x, up.y, up.z );
        break;
    ...
}
```

float  y = EARTH_RADIUS_MILES * sinf( 44.57°);  
float xz = cosf( 44.57°);  
float  x = EARTH_RADIUS_MILES * xz * cosf( 123.27°);  
float  z = EARTH_RADIUS_MILES * xz * sinf( 123.27°);
Note: You Can Also Use these Matrices to Draw the Objects in the Proper Locations instead of using glRotatef( ) and glTranslatef( )

```cpp
 glm::mat4 earth = MakeEarthMatrix( );
 glm::mat4 moon = MakeMoonMatrix( );

glEnable(GL_TEXTURE_2D);

glPushMatrix( );
    glBindTexture(GL_TEXTURE_2D, EarthTex );
    glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE );
    glMultMatrixf( glm::value_ptr(earth) );
glCallList( EarthList );
glPopMatrix( );

glPushMatrix( );
    glBindTexture( GL_TEXTURE_2D, MoonTex );
    glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE );
    glMultMatrixf( glm::value_ptr(moon) );
glCallList( MoonList );
glPopMatrix( );
```