Putting the Eye Position on an Orbiting Body

Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu

Program Setup

```
#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>

const float         ZNEAR =      1.0;
const float         ZFAR = 1000000.0;

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

float               Time;
enum views          WhichView;
```

These are the near and far clipping plane distances in front of the eye. The ZNEAR is typically pretty small, but the ZFAR depends on the scene.

For a lot of our projects, a ZFAR of 1000. works well. But, for something bigger, like the solar system, you will need a ZFAR that will contain the whole depth of the scene.

We are defining them here because we will also need to know ZNEAR to help set the on-the-planet views.

The different eye positions to use

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:

```
const int MAXIMUM_TIME_SECONDS        =   10*60;                 // I decided to use 10 minutes
const int MAXIMUM_TIME_MILLISECONDS   =   1000* MAXIMUM_TIME_SECONDS;
const float ONE_FULL_TURN             =   2. * M_PI;             // this is 2π instead of 360° because glm uses radians
```

In the Animate( ) function:

```
        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:

```
        . . .
        glutIdleFunc( Animate );
        . . .
```



```
void
LatLngToXYZ(float lat, float lng, float rad, glm::vec3* xyzp)
{
        lat  = glm::radians(lat);
        lng = glm::radians(lng);

        xyzp->y = rad * sin(lat);
        float xz = cos(lat);
        xyzp->x = rad * xz * cos(lng);
        xyzp->z = rad * xz * sin(lng);
}
```

Convert a latitude and longitude (in *degrees*) and a planet radius to an (x,y,z)

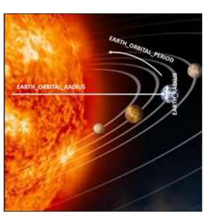
Convert a latitude and longitude eye position and a latitude/longitude look-at position (in *degrees*) and a planet radius to an eye position and a look-at position. A line from the eye position to the look-at position will end up being tangent to the globe.

```
void
SetViewingFromLatLng(float eyeLat, float eyeLng, float lookLat, float lookLng, float rad,   glm::vec4 * eyep, glm::vec4 * lookp)
{
        glm::vec3 eye, look;
        LatLngToXYZ(eyeLat,  eyeLng,  rad,   &eye );
        LatLngToXYZ(lookLat,  lookLng, rad,  &look );
        glm::vec3 up = glm::normalize( eye );           // only true for spheres !!

        glm::vec3 eyeToLook = look - eye;
        glm::vec3 parallelToUp = glm::dot(up, eyeToLook) * eyeToLook;
        eyeToLook = eyeToLook - parallelToUp;

        *eyep  = glm::vec4( eye, 1. );
        *lookp = glm::vec4( eye + eyeToLook, 1. );
}
```

Gram-Schmidt orthogonalization: forces the eyeToLook vector to be perpendicular to the up vector by subtracting the part that is not perpendicular

Physical Parameter Setup

At the top of the program:



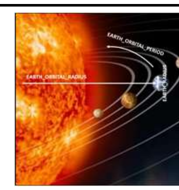
https://www.livescience.com/our-solar-system.html

```
const float SUN_RADIUS_MILES = 432690.;
const float SUN_SPIN_TIME_DAYS_EQUATOR = 25.;
const float SUN_SPIN_TIME_HOURS_EQUATOR = SUN_SPIN_TIME_DAYS_EQUATOR * 24.;
const float SUN_SPIN_TIME_SECONDS_EQUATOR = SUN_SPIN_TIME_HOURS_EQUATOR *60. * 60.;
const float SUN_SPIN_TIME_DAYS_POLES = 35.;
const float SUN_SPIN_TIME_HOURS_POLES = SUN_SPIN_TIME_DAYS_POLES * 24.;
const float SUN_SPIN_TIME_SECONDS_POLES = SUN_SPIN_TIME_HOURS_POLES *60. * 60.;

const float EARTH_RADIUS_MILES = 3964.19;
const float EARTH_ORBITAL_RADIUS_MILES = 92900000.;
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!**

Also, you might need to change the *znear* and *zfar* values in your call to gluPerspective( ) to work with whatever scale you choose.

Possibly Adjusting the Viewing Volume



Remember these lines from our sample code?

```
if( WhichProjection == ORTHO )
        glOrtho( -3., 3.,      -3., 3.,     ZNEAR, ZFAR );
else
        gluPerspective( 90., 1.,            ZNEAR, ZFAR );
```

Be careful because objects can disappear due to *clipping*:

- Items in your scene closer to you than *ZNEAR* in front of your eye will be *clipped away*.
- Items in your scene farther from you than *ZFAR* in front of your eye will be *clipped away*.

This makes them hard to debug. ☺

When we started doing computer graphics, the objects were fairly small, so "**0.1, 1000.**" worked. However, now we are doing solar systems, which could, potentially, have much larger coordinates. So, depending on how you construct your scene, you might have to adjust *ZNEAR* and (especially) *ZFAR*.

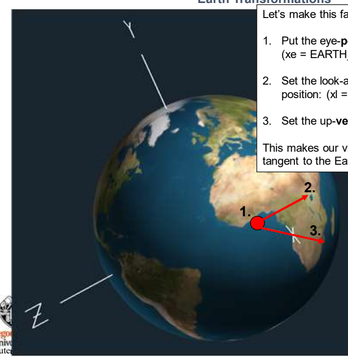
Your Standard Outside View – the One You've Been Using Since Week #0

Put this in the Display( ) function:

```
void
Display( )
{
        . . .
        glm::mat4 m;
        glm::vec4  eye = glm::vec4( 0., 0., 0., 1. );       // a position
        glm::vec4 look = glm::vec4( 0., 0., 0., 1. );       // a position
        glm::vec4   up = glm::vec4( 0., 0., 0., 0. );       // a vector, so doesn't get translations applied

        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;
            . . .
        }
```

← This switch statement is going to switch between four different ways of setting gluLookAt( ), the first is our usual look-at and the rest are for planetary views

Earth Transformations



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 = -100.)
3. Set the up-**vector** to be: (xu = 100., 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

} [$M_{e/s}$]

glCallList(EarthList);

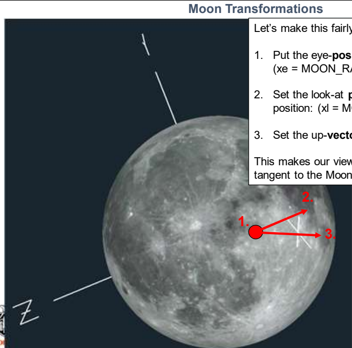
```
glm::mat4
MakeEarthMatrix( )
{
        float earthSpinAngle = Time * EARTH_SPIN_TIME_SECONDS    * ONE_FULL_TURN;
        float earthOrbitAngle = Time * EARTH_ORBIT_TIME_SECONDS * ONE_FULL_TURN;
        glm::mat4 identity = glm::mat4( 1. );
        glm::vec3 yaxis    = glm::vec3( 0., 1., 0. );

/* 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 );     [ Me/s ]

        return erorbity * etransx * erspiny;          // 3 * 2 * 1
}
```


Earth Transformations

Put this in the Display( ) function:

```
void
Display( )
{       . . .
        glm::mat4 e, m;
        glm::vec4  eyePos = glm::vec4( 0., 0., 0., 1. );
        glm::vec4 lookPos = glm::vec4( 0., 0., 0., 1. );
        glm::vec4   upVec = glm::vec4( 0., 0., 0., 0. );  // vectors don't get translations
        glMatrixMode( GL_MODELVIEW );
        glLoadIdentity( );
        switch( WhichView )
        {
            . . .
            case EARTHVIEW:                     // 2nd way to set gluLookAt( )
                e = MakeEarthMatrix( );

                SetViewingFromLatLng(0., 0.,  0., -10., EARTH_RADIUS_MILES, &eyePos, &lookPos);
                e = MakeEarthMatrix();
                upVec = glm::normalize(glm::vec3(eyePos));
                eyePos  = e * eyePos;
                lookPos = e * lookPos;
                upVec   = glm::vec3(  e * glm::vec4( upVec, 0. )  );

                glTranslatef(0., 0., -6.f * ZNEAR);
                gluLookAt( eyePos.x, eyePos.y, eyePos.z,   lookPos.x, lookPos.y, lookPos.z,   upVec.x, upVec.y, upVec.z );
                break;
            . . .
        }
```


Moon Transformations



Let's make this fairly straightforward. In model coordinates:

1. Put the eye-**position** at the corner of its Equator and Prime Meridian (xe = MOON_RADIUS_MILES, ye = 0., ze = 0.)
2. Set the look-at **position** to be on a straight-line east of the eye-position: (xl = MOON_RADIUS_MILES, yl = 0., zl = -100.)
3. Set the up-**vector** to be: (xu = 100.,yu = 0., zu = 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

} [$M_{m/e}$]

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

} [$M_{e/s}$]

glCallList(MoonList);

```
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      [ Me/s ] * [ Mm/e ]
}
```

Note that *EarthSpinAngle* has no effect on the Moon's matrix

Moon Transformations

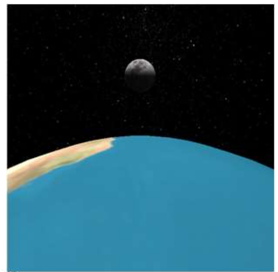
13

Put this in the Display( ) function:

```
void
Display( )
{		. . .
		glm::mat4 e, m;
		glm::vec4  eyePos = glm::vec4( 0., 0., 0., 1. );
		glm::vec4 lookPos = glm::vec4( 0., 0., 0., 1. );
		glm::vec4   upVec = glm::vec4( 0., 0., 0., 0. );  // vectors don't get translations

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

				SetViewingFromLatLng(0., 0., 0., -10., MOON_RADIUS_MILES, &eyePos, &lookPos);
				m = MakeMoonMatrix();
				upVec = glm::normalize(glm::vec3(eyePos));
				eyePos  = m * eyePos;
				lookPos = m * lookPos;
				upVec = glm::vec3(m * glm::vec4(upVec, 0.));

				glTranslatef(0., 0., -4.f * ZNEAR);
				gluLookAt( eyePos.x, eyePos.y, eyePos.z,   lookPos.x, lookPos.y, lookPos.z,   upVec.x, upVec.y, upVec.z );
				break;
			. . .
		}
```


Transformations In Action!

14



Images by Christopher Weiner

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

15

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:

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

Then multiply **x**, **y**, and **z** by EARTH_RADIUS_MILES

Earth/Corvallis Transformations

16

```
float  y  = EARTH_RADIUS_MILES  * sin( 44.57°);
float xz = cos( 44.57° );
float  x =  EARTH_RADIUS_MILES  * xz * cos( 123.27° );
float  z =  EARTH_RADIUS_MILES  * xz * sin(  123.27° );
```

Put this in the Display( ) function:

```
void
Display( )
{		. . .
		glm::mat4 e, m;
		glm::vec4  eyePos = glm::vec4( 0., 0., 0., 1. );
		glm::vec4 lookPos = 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 CORVALLISVIEW:		// 4th way to set gluLookAt( )
				SetViewingFromLatLng(44.57f, 123.27f, 44.57f, 122.27f, EARTH_RADIUS_MILES, &eyePos, &lookPos);
				e = MakeEarthMatrix();
				upVec = glm::normalize(glm::vec3(eyePos));
				eyePos =  e * eyePos;
				lookPos = e * lookPos;
				upVec = glm::vec3(e * glm::vec4(upVec, 0.));

				glTranslatef(0., 0., -6.f * ZNEAR);
				gluLookAt( eyePos.x, eyePos.y, eyePos.z,   lookPos.x, lookPos.y, lookPos.z,   upVec.x, upVec.y, upVec.z );
				break;
			. . .
		}
```


Note: You Can Also Use these Matrices to Draw the Objects in the Proper Locations instead of using glRotatef( ) and glTranslatef( )

17

```
glm::mat4  e  = MakeEarthMatrix( );
glm::mat4  m = 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(e)   );
		glCallList( EarthList );
glPopMatrix( );

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