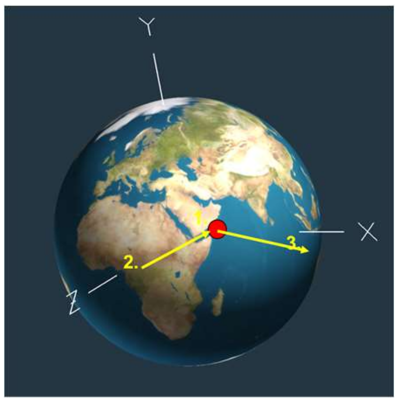
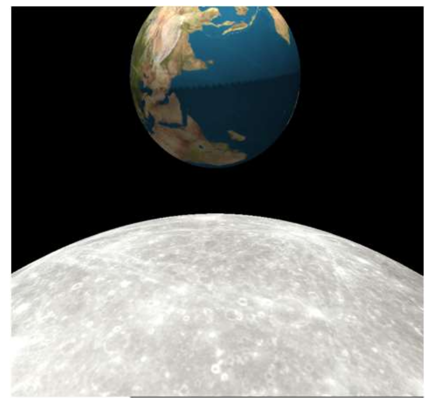
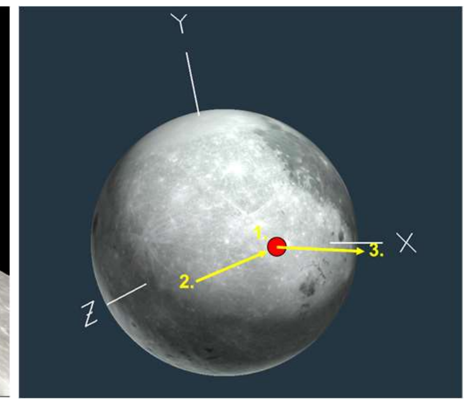
Placing the Eye Position on an Orbiting Body



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 "glm/gtc/type_ptr.hpp"

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

const float          EYEDISTFACTOR = 0.75f;
const float          ZFAR = 1000000.0f;
const float          FOVDEG =     90.0f;

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

float                Time;
```

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. worked well. But, for something bigger, like the solar system, you will need a ZFAR that will contain the whole depth of the scene.

The look-at position is on the planet. The eye-position is behind that on a line tangent to the planet. The distance from the look-at position to the eye-position will depend on this factor times the radius of the specific planet. 0.75 – 1.25 worked well for me.

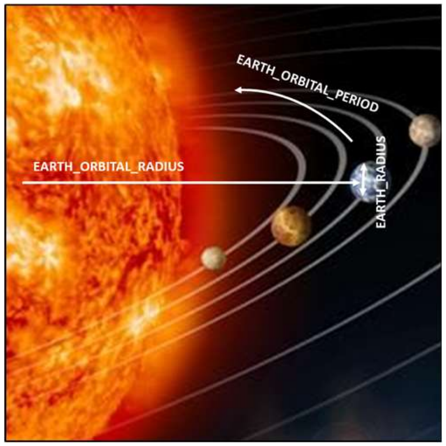
Perspective field-of-view angle (degrees)

The different eye views you can use

Which eye view is the current one

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

Possibly Adjusting the Viewing Volume



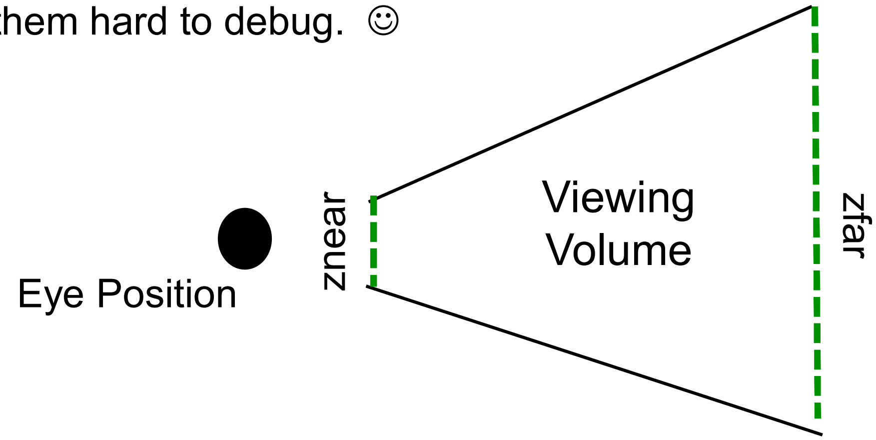
Remember these lines from our sample code?

```
if( WhichProjection == ORTHO )
        glOrtho( -3., 3.,     -3., 3.,     znear, zfar );
else
        gluPerspective( FOVDEG, 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. ☺

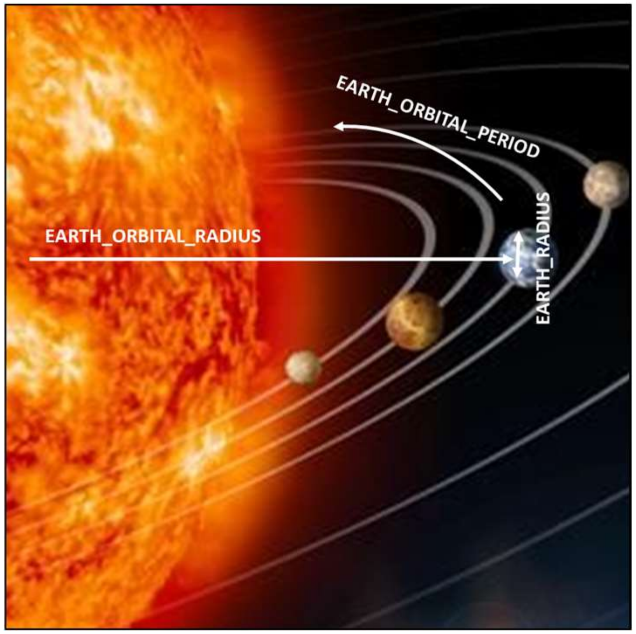


When we started doing computer graphics, the objects were fairly small, so "**0.1f, 1000.f**" worked. However, now we are doing solar systems, which could, potentially, have much larger coordinates. Our special eye-position math helps set *znear*. But, depending on how you construct your scene, you might have to adjust *zfar*.

Physical Parameter Setup

At the top of the program:

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

const float EARTH_RADIUS_MILES = 3964.19f;
const float EARTH_TILT_ANGLE_DEG = 23.44f;
const float EARTH_ORBITAL_RADIUS_MILES = 92900000.f;
const float EARTH_ORBIT_TIME_DAYS = 365.3f;
const float EARTH_ORBIT_TIME_HOURS = EARTH_ORBIT_TIME_DAYS  * 24.f;
const float EARTH_ORBIT_TIME_SECONDS = EARTH_ORBIT_TIME_HOURS * 60.f * 60.f;
const float EARTH_SPIN_TIME_DAYS = 0.9971f;
const float EARTH_SPIN_TIME_HOURS = EARTH_SPIN_TIME_DAYS * 24.f;
const float EARTH_SPIN_TIME_SECONDS = EARTH_SPIN_TIME_HOURS * 60.f * 60.f;

const float MOON_RADIUS_MILES = 1079.6f;
const float MOON_ORBITAL_RADIUS_MILES = 238900.f;
const float MOON_ORBIT_TIME_DAYS = 27.3f;
const float MOON_ORBIT_TIME_HOURS = MOON_ORBIT_TIME_DAYS * 24.f;
const float MOON_ORBIT_TIME_SECONDS = MOON_ORBIT_TIME_HOURS * 60.f * 60.f;
const float MOON_SPIN_TIME_DAYS = MOON_ORBIT_TIME_DAYS;
const float MOON_SPIN_TIME_HOURS = MOON_SPIN_TIME_DAYS * 24.f;
const float MOON_SPIN_TIME_SECONDS = MOON_SPIN_TIME_HOURS * 60.f * 60.f;
```



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

***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.

Here's How I Tried Scaling Things


```
// time scale wrt days (bigger creates slower):
const float TS = 1.08f / ( 24.f * 60.f * 60.f );
// this creates a 1 second on the screen = 1 day in the real solar system

// planet radius scale:
const float PRS = 0.1f / 3964.f;

// sun radius scale:
const float SRS = PRS/40.f;

// earth orbital radius scale:
const float EORS = 1.f / 93000000.f;

// moon orbital radius scale:
const float MORS = 150.f * EORS;

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

const float EARTH_RADIUS_MILES = PRS*3964.19f;
const float EARTH_TILT_ANGLE_DEG = 23.44f;
const float EARTH_ORBITAL_RADIUS_MILES = EORS*92900000.f;
const float EARTH_ORBIT_TIME_DAYS = TS*365.3f;
const float EARTH_ORBIT_TIME_HOURS = EARTH_ORBIT_TIME_DAYS  * 24.f;
const float EARTH_ORBIT_TIME_SECONDS = EARTH_ORBIT_TIME_HOURS * 60.f * 60.f;
const float EARTH_SPIN_TIME_DAYS = TS*0.9971f;
const float EARTH_SPIN_TIME_HOURS = EARTH_SPIN_TIME_DAYS * 24.f;
const float EARTH_SPIN_TIME_SECONDS = EARTH_SPIN_TIME_HOURS * 60.f * 60.f;

const float MOON_RADIUS_MILES = PRS*1079.6f;
const float MOON_ORBITAL_RADIUS_MILES = MORS*238900.f;
const float MOON_ORBITAL_INCLINATION_DEG = 5.14f;
const float MOON_ORBIT_TIME_DAYS = TS*27.3f;
const float MOON_ORBIT_TIME_HOURS = MOON_ORBIT_TIME_DAYS * 24.f;
const float MOON_ORBIT_TIME_SECONDS = MOON_ORBIT_TIME_HOURS * 60.f * 60.f;
const float MOON_SPIN_TIME_DAYS = MOON_ORBIT_TIME_DAYS;
const float MOON_SPIN_TIME_HOURS = MOON_SPIN_TIME_DAYS * 24.f;
const float MOON_SPIN_TIME_SECONDS = MOON_SPIN_TIME_HOURS * 60.f * 60.f;
```

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.f * F_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:

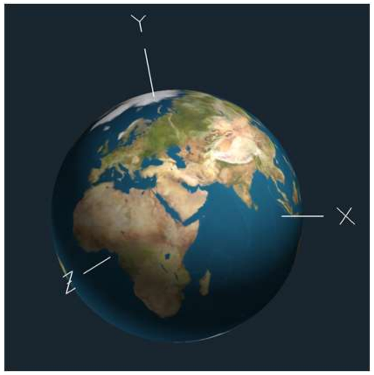
```
        . . .
        glutIdleFunc( Animate );
        . . .
```

Display List Setup


```
SphereDL = glGenLists( 1 );
glNewList( SphereDL, GL_COMPILE );
        OsuSphere( 1.f, 512, 512 );
glEndList( );

SunDL = glGenLists( 1 );
glNewList( SunDL, GL_COMPILE );
        glBindTexture(GL_TEXTURE_2D, SunTexture );
        glPushMatrix();
            glScalef( SUN_RADIUS_MILES, SUN_RADIUS_MILES, SUN_RADIUS_MILES);
            glCallList(SphereDL);
        glPopMatrix();
glEndList( );

EarthDL = glGenLists( 1 );
glNewList( EarthDL, GL_COMPILE );
        glBindTexture(GL_TEXTURE_2D, EarthTexture );
        glPushMatrix();
            glScalef( EARTH_RADIUS_MILES, EARTH_RADIUS_MILES, EARTH_RADIUS_MILES);
            glCallList(SphereDL);
        glPopMatrix();
glEndList( );

MoonDL = glGenLists( 1 );
glNewList( MoonDL, GL_COMPILE );
        glBindTexture(GL_TEXTURE_2D, MoonTexture );
        glPushMatrix();
            glScalef(MOON_RADIUS_MILES, MOON_RADIUS_MILES, MOON_RADIUS_MILES);
            glCallList(SphereDL);
        glPopMatrix();
glEndList( );
```

A display list can call a previously-created display list

Earth Transformations



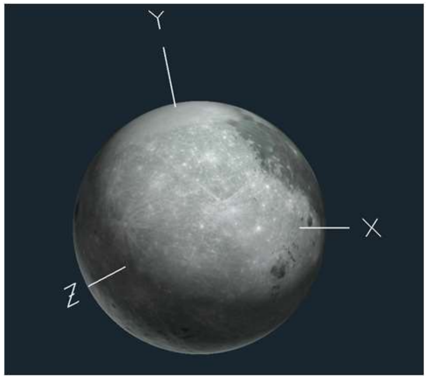
Steps to transform the Earth-eye-viewing system into Solar System Coordinates:
Using OsuSphere( ) draw the Earth into a display list at (0.,0.,0.), i.e., the Sun's center

1. Spin the Earth by *EarthSpinAngle* about its Y axis
2. Tilt the Earth by 23.5° about its X axis
3. Translate the Earth to where it belongs in its orbit

[$M_{e/s}$]

glCallList(EarthList);

```
glm::mat4
MakeEarthMatrix( )
{
	const float earthSpinAngle  =  ONE_FULL_TURN * Time * TimeScale / EARTH_SPIN_TIME_SECONDS;
	const float earthOrbitAngle =  ONE_FULL_TURN * Time * TimeScale / EARTH_ORBIT_TIME_SECONDS;
	const glm::mat4 identity = glm::mat4( 1. );
	const glm::vec3 yaxis    = glm::vec3( 0., 1., 0. );
	const glm::vec3 zaxis    = glm::vec3( 0., 0., 1. );
	// note: because the earth is tilted, we cannot play the translate+rotate trick -- the tilt must always be in the same direction
	// instead we must translate the earth to where it really belongs:
	glm::vec3 eorbitPosition = glm::vec3( EARTH_ORBITAL_RADIUS_MILES * cosf(earthOrbitAngle), 0., -EARTH_ORBITAL_RADIUS_MILES * sinf(earthOrbitAngle) );

/* 3. */	glm::mat4 etransx=   glm::translate( identity, eorbitPosition);
/* 2. */	glm::mat4 etilt    =   glm::rotate(     identity, glm::radians(EARTH_TILT_ANGLE_DEG), zaxis );
/* 1. */	glm::mat4 erspiny   = glm::rotate(     identity, earthSpinAngle - earthOrbitAngle, yaxis );// the orbit angle also does the spin

	return etransx * etilt * erspiny;// 3 * 2 * 1  [ Me/s ]
}
```



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. Incline the Moon by *MOON_ORBITAL_INCLINATION_DEG* about the Z axis
5. Translate the Earth by *EARTH_ORBITAL_RADIUS_MILES* in its X direction
6. Revolve the Earth by *EarthOrbitAngle* about the Sun's Y axis

glCallList(MoonList);

Steps 1–4: **[$M_{m/e}$]**

Steps 5–6: **[$M_{e/s}$]**

Note that *EarthSpinAngle* and *EarthTiltAngle* have no effect on the Moon's matrix

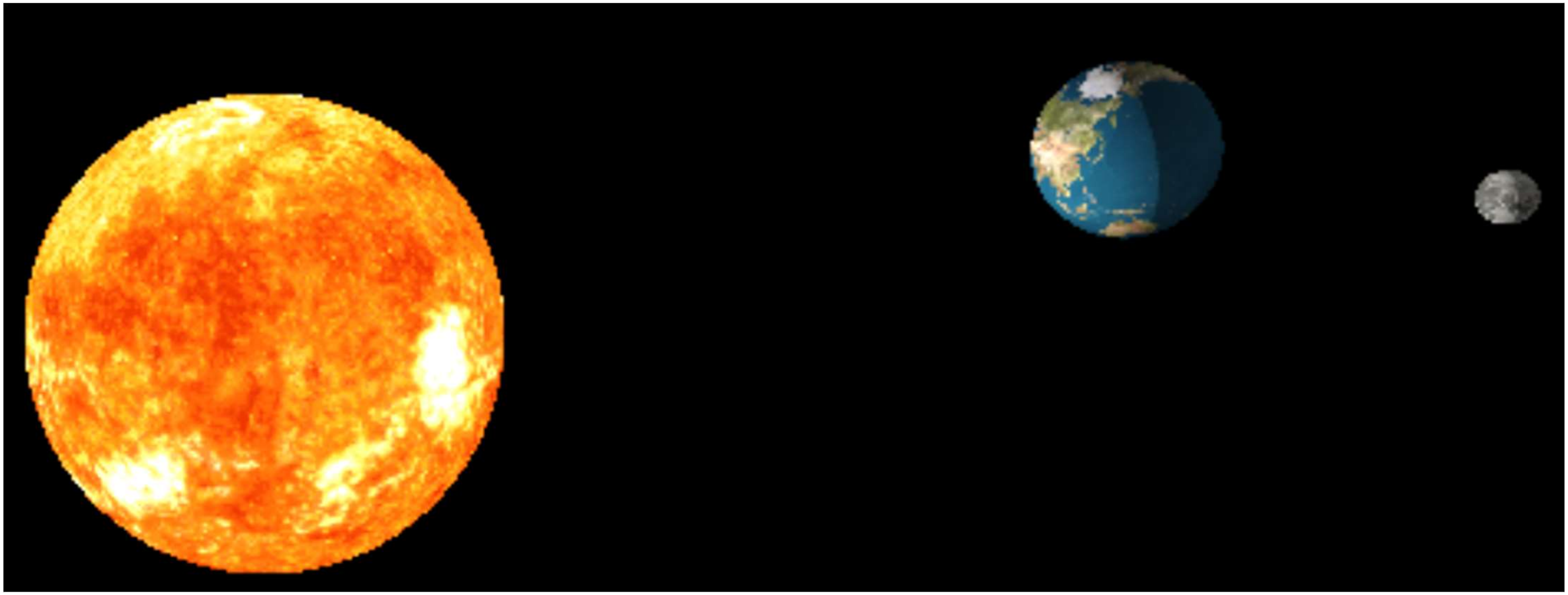
```
glm::mat4
MakeMoonMatrix( )
{
	float moonSpinAngle    = ONE_FULL_TURN * Time * TimeScale / MOON_SPIN_TIME_SECONDS;
	float moonOrbitAngle   = ONE_FULL_TURN * Time * TimeScale / MOON_ORBIT_TIME_SECONDS;
	float earthOrbitAngle  = ONE_FULL_TURN * Time * TimeScale / EARTH_ORBIT_TIME_SECONDS;
	glm::mat4 identity = glm::mat4( 1. );
	glm::vec3 yaxis    = glm::vec3( 0., 1., 0. );
	glm::vec3 zaxis    = glm::vec3(0., 0., 1.);

/* 6. */	glm::mat4 erorbity    = glm::rotate(     identity, earthOrbitAngle, yaxis );
/* 5. */	glm::mat4 etransx     = glm::translate(  identity, glm::vec3( EARTH_ORBITAL_RADIUS_MILES, 0., 0. ) );
/* 4. */	glm::mat4 mrinclinez = glm::rotate(     identity, glm::radians(MOON_ORBITAL_INCLINATION_DEG), zaxis );
/* 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-moonOrbitAngle, yaxis );		// the orbit angle also does the spin

	return erorbity * etransx * mrinclinez * mrorbity * mtransx * mrspiny;  // 6 * 5 * 4 * 3 * 2 * 1}
```

[$M_{e/s}$] * [$M_{m/e}$]

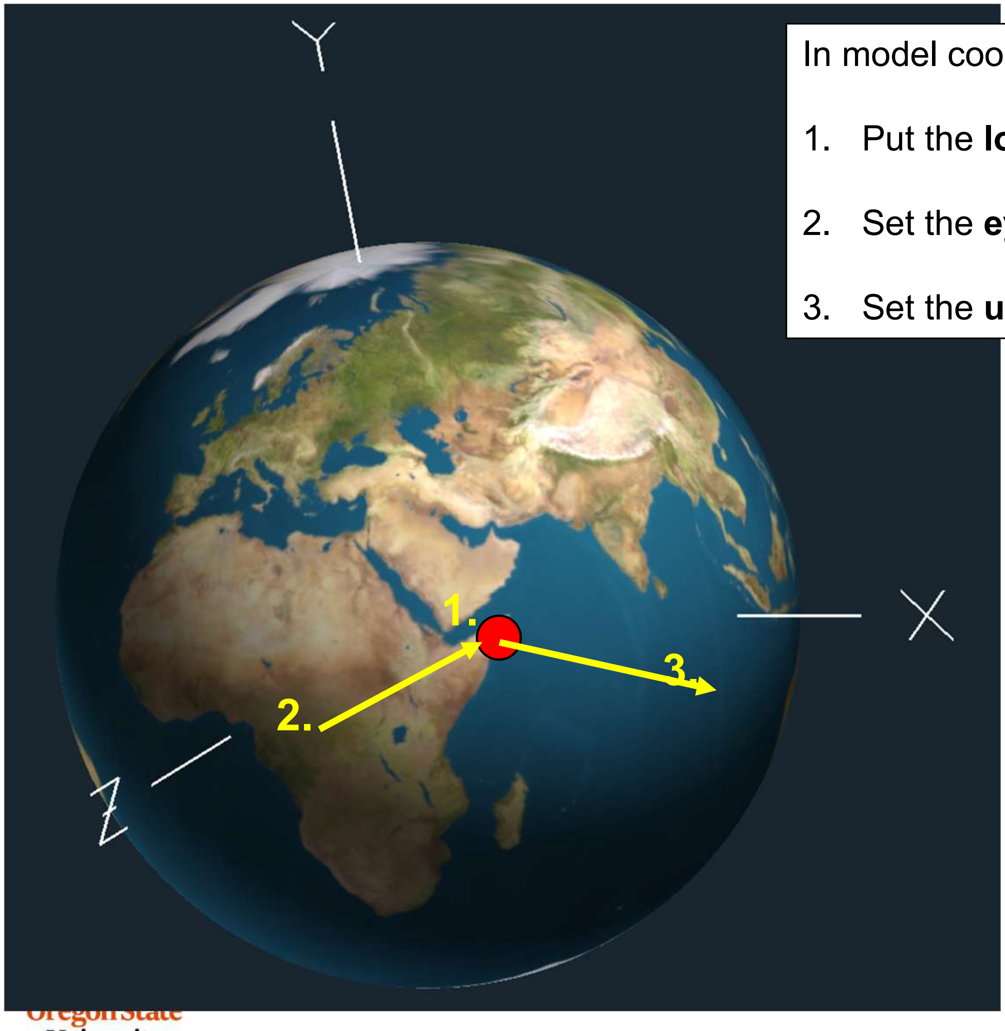
Note: You Can Use these Matrices to Draw the Objects in the Proper Locations instead of using glRotatef( ) and glTranslatef( )

```
glm::mat4  e  = MakeEarthMatrix( );
glm::mat4  m = MakeMoonMatrix( );

glEnable(GL_TEXTURE_2D);
If( DoingLighting )
{
        glEnable( GL_LIGHTING );
        glEnable( GL_LIGHT0 );
}

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( );
```

Transformations In Action!



Earth Viewing



In model coordinates:

1. Put the **look-at-position** at an arbitrary latitude and longitude
2. Set the **eye-position** at an arbitrary latitude and longitude
3. Set the **up-vector** to be perpendicular to the surface at the look-at position

Now, all we have to do is transform those two locations and one vector into Solar System Coordinates (I hate to call them "World Coordinates" here…).

Converting Latitude-Longitude to XYZ

```
glm::vec3
LatLngToXYZ(float lat, float lng, float rad )
{
        lat   = glm::radians(lat);
        lng  = glm::radians(lng);
        glm::vec3 xyz;
        float xz = cosf(lat);

        xyz.x = -rad * xz * cosf(lng);
        xyz.y =  rad * sinf(lat);
        xyz.z =  rad * xz * sinf(lng);
        return xyz;
}
```

Convert a latitude and longitude (in *degrees*) and a planet radius to an (x,y,z). This assumes that (0,0,0) is at the center of the planet.

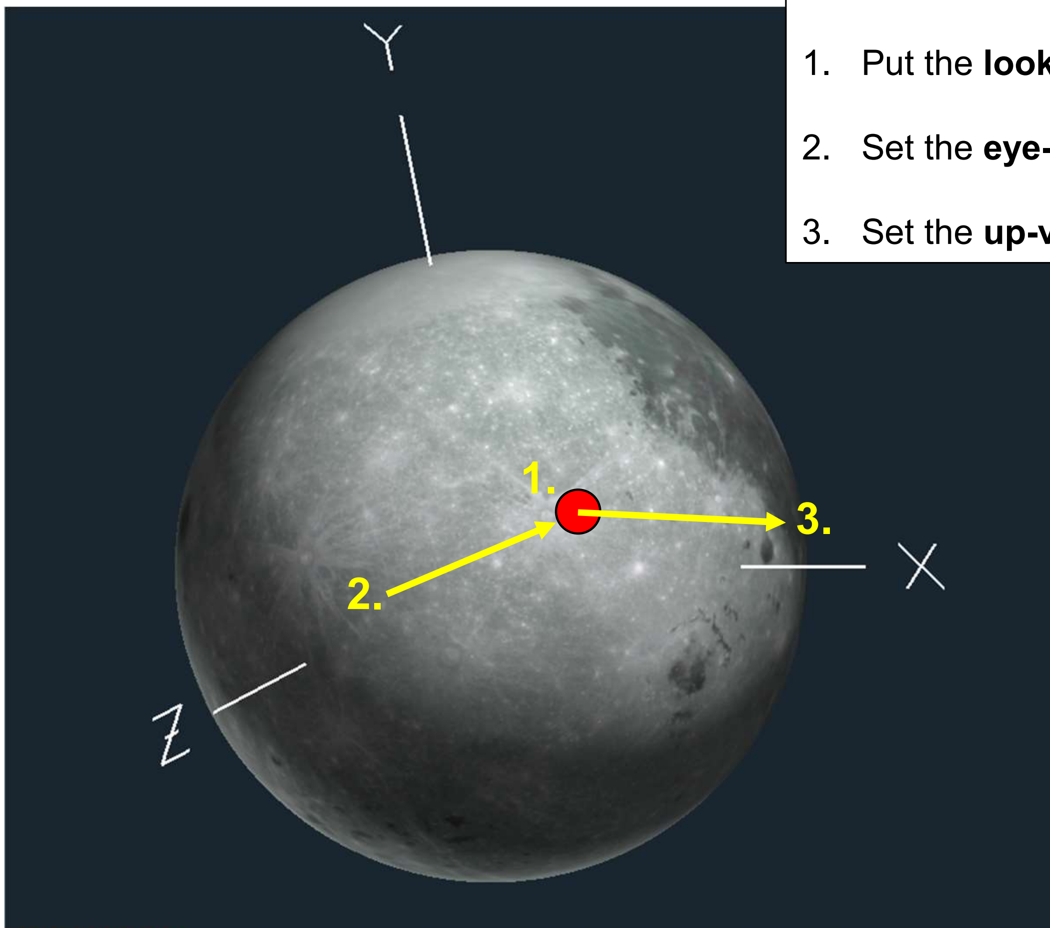
Moon Viewing

In model coordinates:

1. Put the **look-at-position** at an arbitrary latitude and longitude

2. Set the **eye-position** at an arbitrary latitude and longitude

3. Set the **up-vector** to be perpendicular to the surface at the look-at position



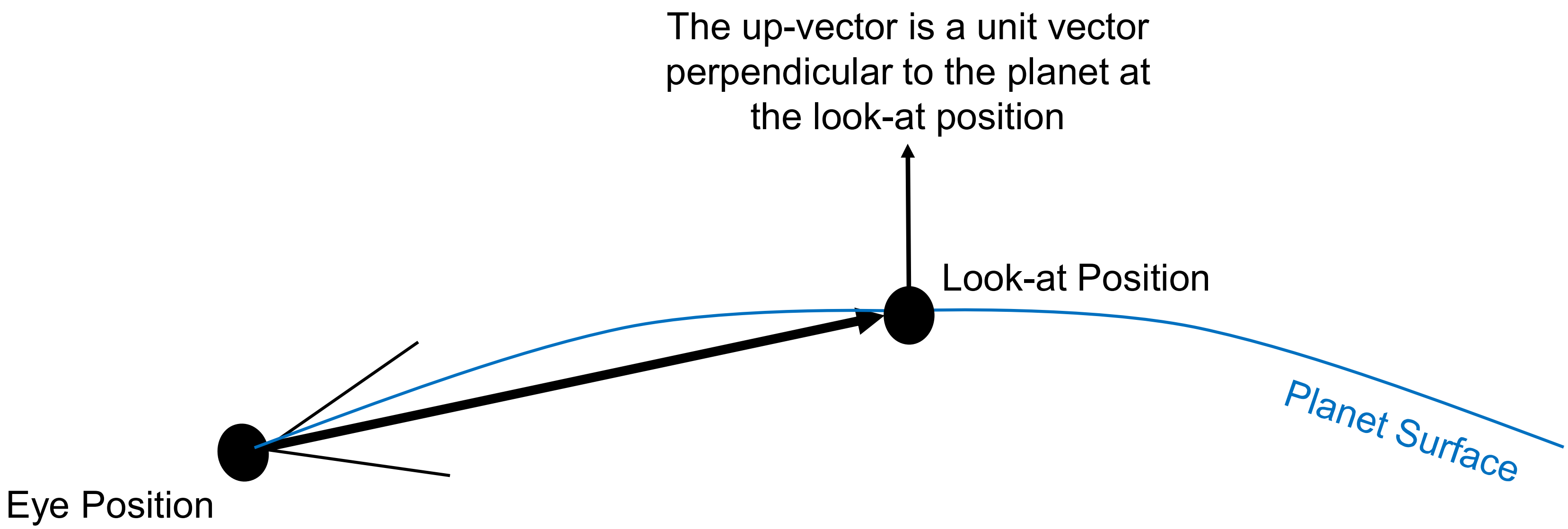
Now, all we have to do is transform those two locations and one vector into Solar System Coordinates (I hate to call them “World Coordinates” here…).

Converting Eye+Look Latitude-Longitude to Eye+Look XYZ:
It would be nice if this was all there was to it. Hint: there's more!

```
void
SetViewingFromLatLng(float eyeLat, float eyeLng, float lookLat, float lookLng, float rad,   glm::vec4 * eyep, glm::vec4 * lookp, glm::vec4 * upp, float *znearp )
{
	glm::vec3 center = glm::vec3( 0., 0., 0. );				// center of planet
	glm::vec3 eye  = LatLngToXYZ(eyeLat,  eyeLng,  rad );
	glm::vec3 look = LatLngToXYZ(lookLat, lookLng, rad );
	glm::vec3 upVec = glm::normalize( look - center );			// perpendicular to the globe at the look position

	*eyep  = glm::vec4( eye,	1. );
	*lookp = glm::vec4( look,	1. );
	*upp   = glm::vec4( upVec,	0. );
	*znearp = 0.1f;
}
```

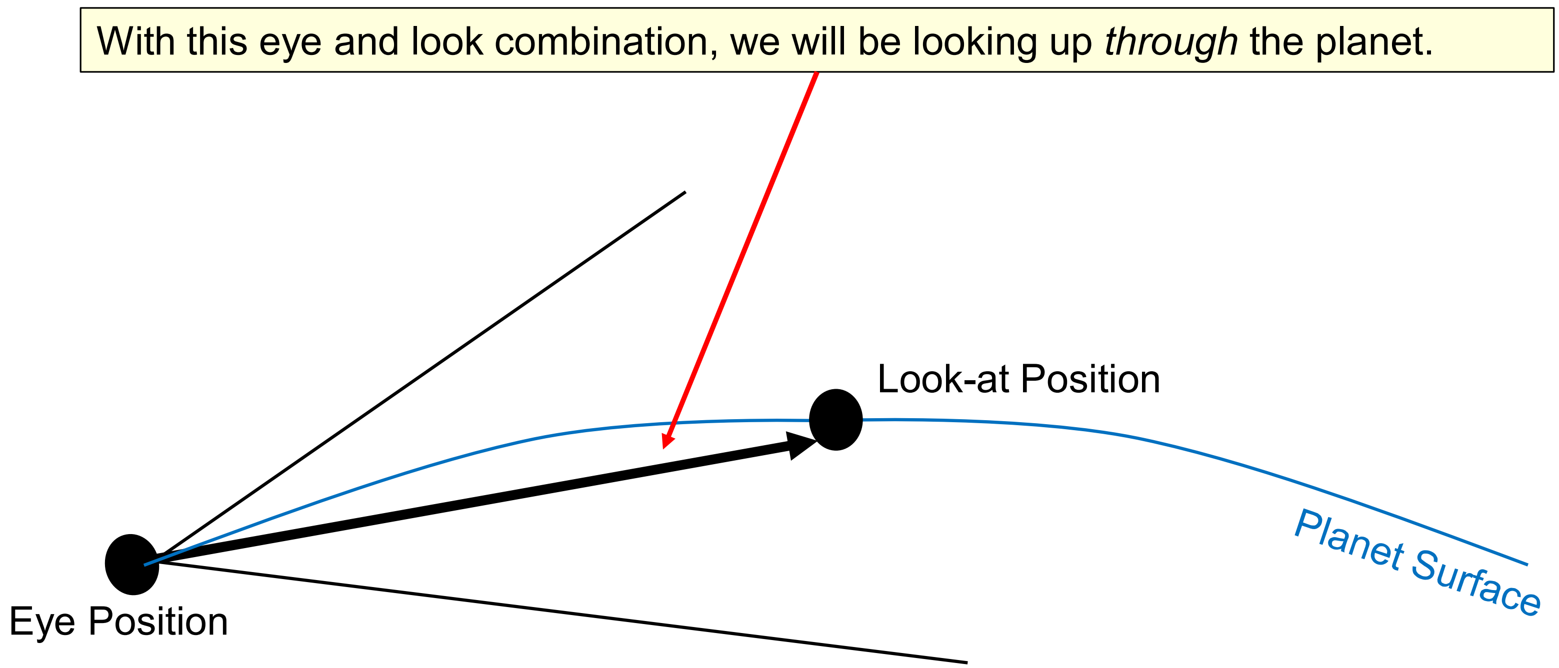
Convert a latitude and longitude eye position and look-at position (in *degrees*) and a planet radius to an eye position, look-at position, up-vector, and near clipping distance in XYZ coordinates.

Here's the Viewing Strategy



Strategy:
1. Pick a (latitude,longitude) for the eye position
2. Pick a (latitude,longitude) for the look-at position
3. Use the surface normal at the look-at position for the up-vector

But There is a Problem -- We Cannot Leave the Eye There!

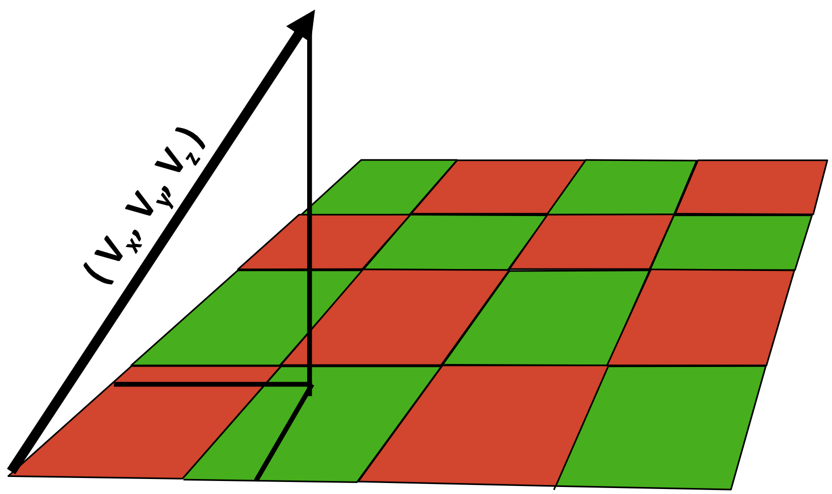


We are about to use some vector math!
Vector math is a *big deal* in computer graphics. Your games use it all the time.
You don’t have to completely understand vector math to appreciate what we are about to use it for.

But, if you would like a review of vector math, go to:

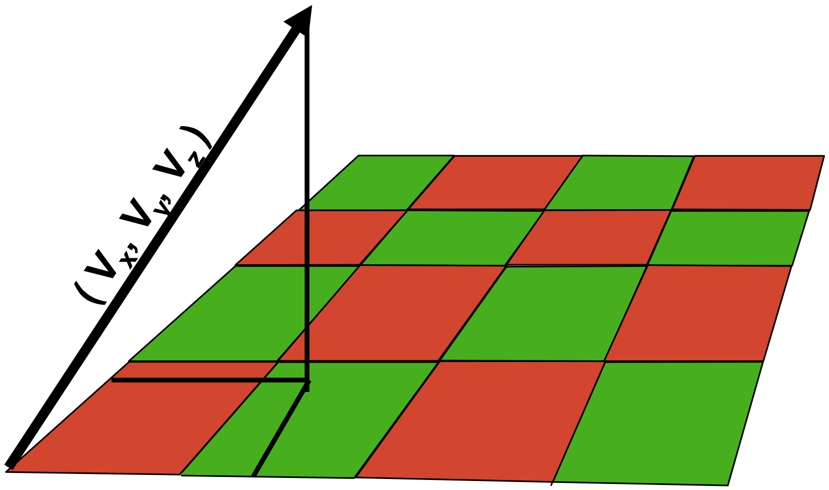
http://cs.oregonstate.edu/~mjb/cs491/Handouts/vectors.1pp.pdf

Sidebar: Vectors have Direction and Magnitude



Magnitude: $$\|V\| = \sqrt{V_x^2 + V_y^2 + V_z^2}$$

Sidebar: Unit Vectors have a Magnitude = 1.0

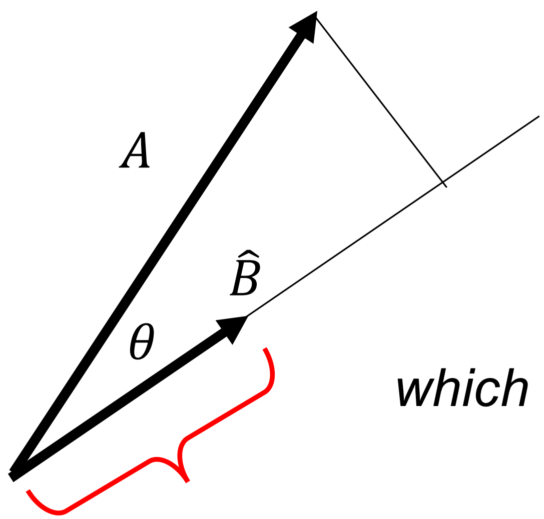


$$\|V\| = \sqrt{V_x^2 + V_y^2 + V_z^2}$$

$$\hat{V} = \frac{V}{\|V\|}$$

The circumflex (^) tells us this is a unit vector

Sidebar:
Using the Vector Dot Product to Determine How Much of Vector-A Lives in the Vector-B Direction



$$A \cdot B = \|A\| \|B\| cos\theta$$

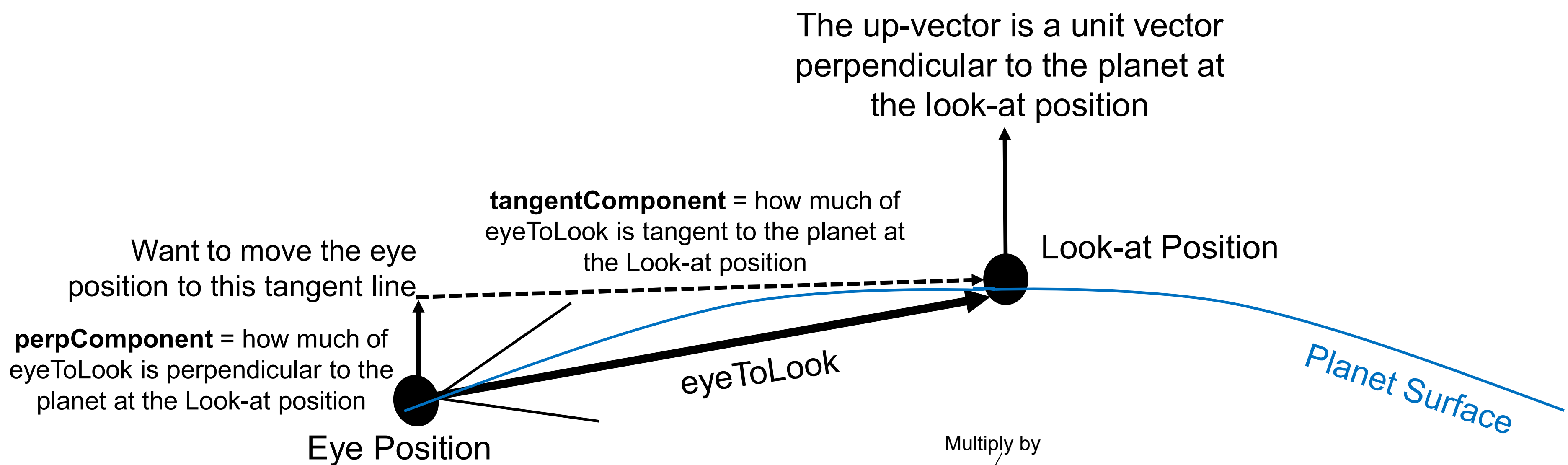
$$A \cdot \hat{B} = \|A\| cos\theta$$

which is the length of the projection of A onto the B line

So, how much of A lives in the $\hat{B}$ direction is that magnitude times the B unit vector:

$$(A \cdot \hat{B})\ \hat{B}$$

The Viewing Strategy is to Make the Eye-to-Look Tangent to the Planet



Multiply by

Vector dot product

$$\overline{\text{perpComponent}} = (\ \overline{\text{eyeToLook}} \cdot \widehat{\text{upvec}}\ ) * \widehat{\text{upVec}}$$

$$\overline{\text{perpComponent}} + \overline{\text{tangentComponent}} = \overline{\text{eyeToLook}}$$

$$\overline{\text{tangentComponent}} = \overline{\text{eyeToLook}} - \overline{\text{perpComponent}}$$

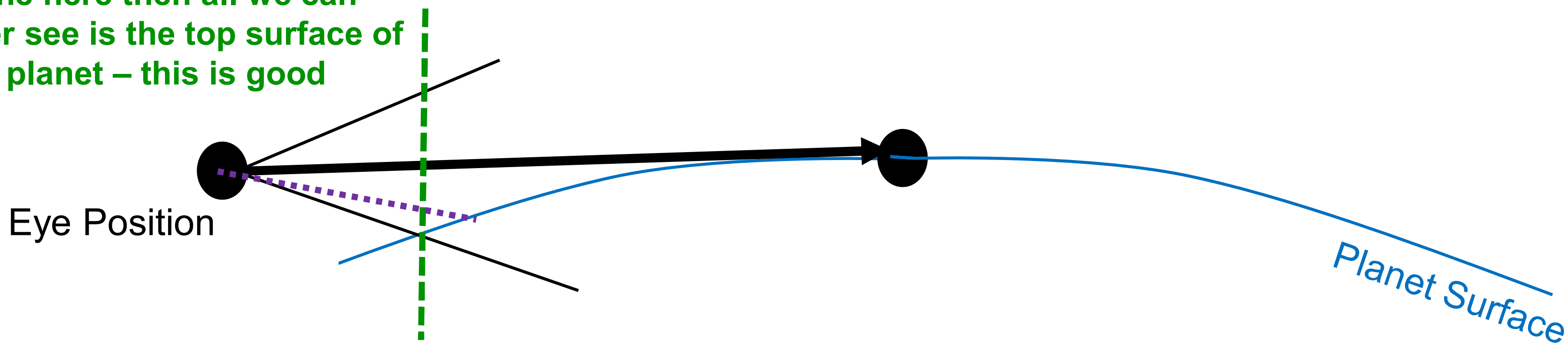
This line is of the form: $(A \cdot \hat{B})\ \hat{B}$

$\overline{\qquad}$ = **a glm::vec3**

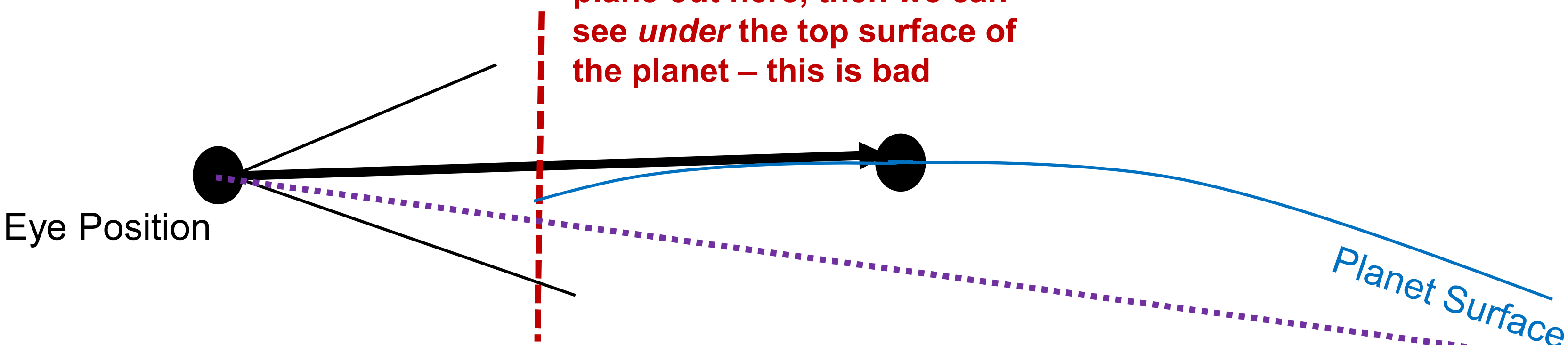
$\wedge$ = **a unitized glm::vec3**

But There is *Another* Problem – We Have to Set the Near Clipping Plane Carefully

If we put the near clipping plane here then all we can ever see is the top surface of the planet – this is good



But if we put the near clipping plane out here, then we can see *under* the top surface of the planet – this is bad

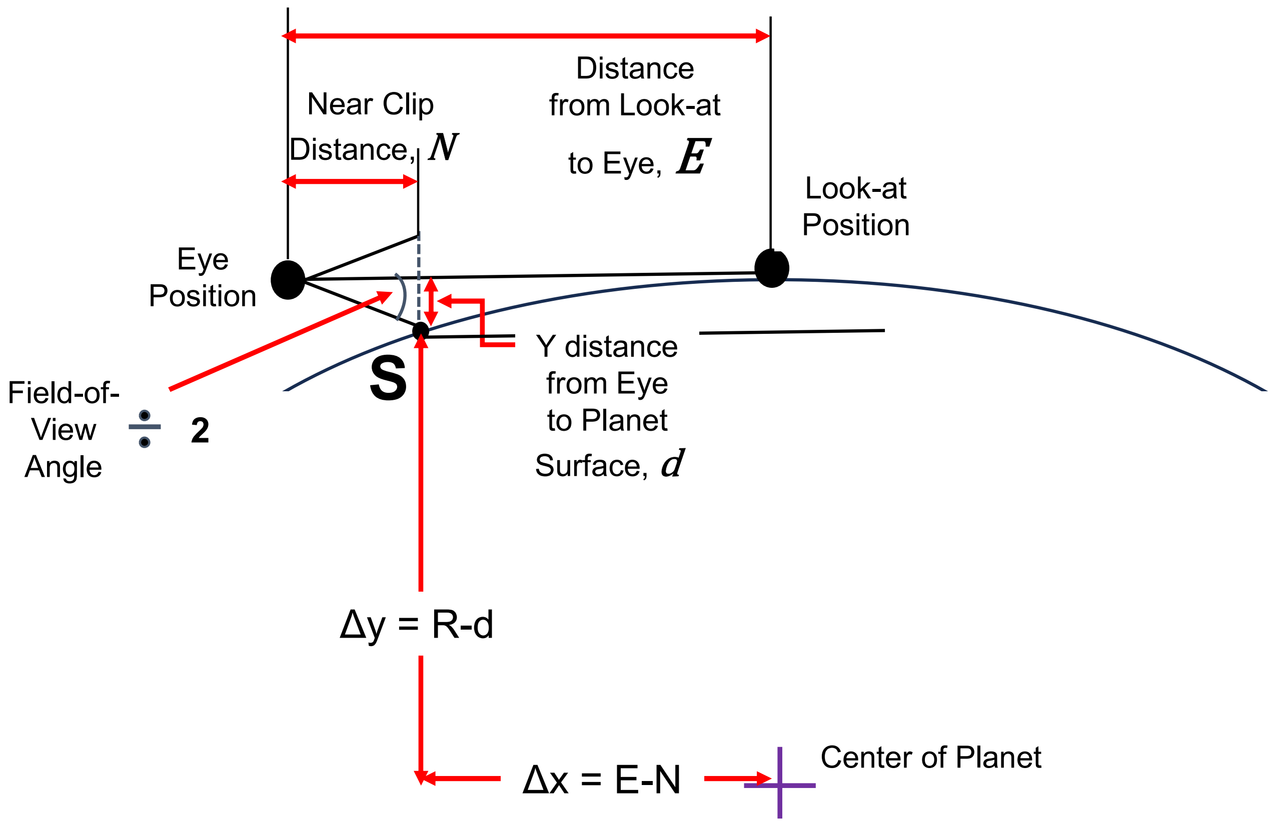


Seeing Under the Top Surface of the Planet Shows the Continents Reversed!

This is not just bad, it is *very* bad

We need to place the eye and the eye's near clipping plane such that no part of what the eye can see is under the planet's surface.

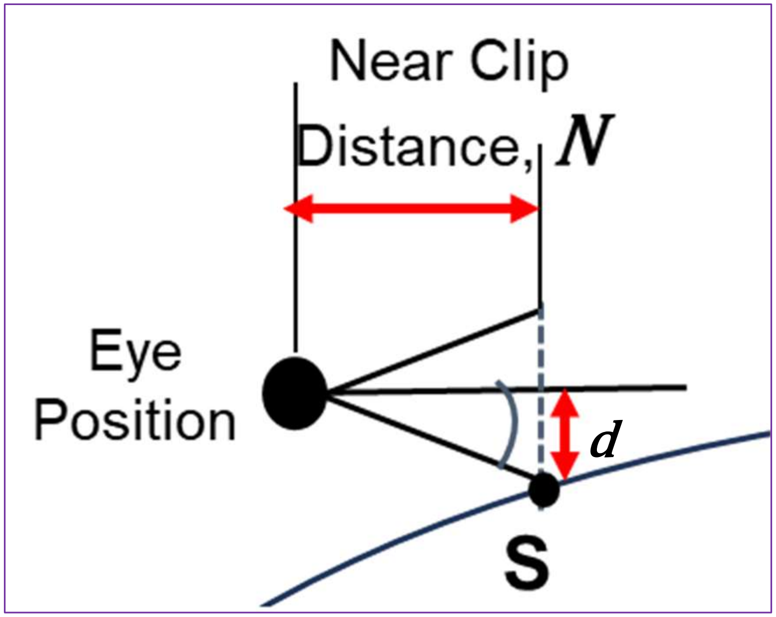
Here's the Viewing Strategy: Use All These Parameters to Compute *N*



At the point **S** where the near clipping plane touches the planet surface:

$$(R-d)^2 + (E-N)^2 = R^2$$

Here's the Viewing Strategy



$$\frac{d}{N} = tan\left(\frac{fov}{2}\right) = "T"$$

$$d = NT$$

At the point S:

$$(R - d)^2 + (E - N)^2 = R^2$$

$$\downarrow$$

$$(R - NT)^2 + (fR - N)^2 = R^2$$

$$\downarrow$$

$$\cancel{R^2} - 2RNT + N^2T^2 + (fR)^2 - 2fRN + N^2 = \cancel{R^2}$$

$$\downarrow$$

$$N^2\left[T^2 + 1\right] + N[-2RT - 2fR] + \left[(fR)^2\right] = 0$$

$$\downarrow$$

$$AN^2 + BN + C = 0$$

$$\downarrow$$

$$N = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

Here's the Viewing Strategy

$$N = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

1. We get a good look-at position from a point on the planet.
2. We get a good eye-position by backing up a distance E from the look-at position on a line tangent to the planet.
3. We get a good value for the distance E by multiplying the planet radius by **EYEDISTFACTOR**
4. We get a good near clipping plane distance by solving the above quadratic equation.
5. Of the two solutions, we take the minimum.

By experimenting, I found decent values for f to be between 0.75 – 1.25
In these notes (slide #2), f is called **EYEDISTFACTOR**.

Converting Eye+Look Latitude-Longitude to Eye+Look XYZ – the Full Detail

Convert a latitude and longitude eye-position and look-at position (in *degrees*) and a planet radius to an eye position and a look-at position in XYZ coordinates. A line from the eye position to the look-at position needs to be forced to be tangent to the globe.

```
void
SetViewingFromLatLng(float eyeLat, float eyeLng, float lookLat, float lookLng, float rad,   glm::vec4 * eyep, glm::vec4 * lookp, glm::vec4 * upp, float *znearp )
{
       const glm::vec3 center = glm::vec3( 0., 0., 0. );// center of planet
       glm::vec3 eye  = LatLngToXYZ(eyeLat,  eyeLng,  rad );
       glm::vec3 look = LatLngToXYZ(lookLat, lookLng, rad );
       glm::vec3 upVec = glm::normalize( look - center );// perpendicular to the globe at the look position
       glm::vec3 eyeToLook = look - eye;// vector from eye to look
       float znear;// how far in front of eye to near clipping plane

       float T = tanf(glm::radians(FOVDEG/2.f));
       float distToEye = EYEDISTFACTOR * rad;                // distance from look to eye'

       glm::vec3 perpComponent = glm::dot( eyeToLook, upVec ) * upVec;
       glm::vec3 tangentComponent = eyeToLook - perpComponent;
       glm::vec3 eyeprime = look  -  distToEye * glm::normalize( tangentComponent );

       float A = SQR(T) + 1.f;
       float B = -2.f*rad*T - 2.f*distToEye;
       float C = SQR(distToEye);
      float disc = sqrtf(SQR(B)-4.f*A*C);
      znear =    ( -B + disc ) / (2.f*A);
      float n2 = ( -B - disc ) / (2.f*A);
      if(n2 < znear)
            znear = n2;

      *eyep  = glm::vec4( eyeprime,  1. );
      *lookp = glm::vec4( look,         1. );
      *upp   = glm::vec4( upVec,       0. );
      *znearp = znear;
}
```

Use a vector dot product to see how much of the eyeToLook vector is perpendicular to the surface. Then subtract that away to see how much of eyeToLook is tangent to the surface, which is what we want.

Standard Outside Viewing

Put this in the Display( ) function:

```
void
Display( )
{
        . . .
        glm::mat4 e  = MakeEarthMatrix( );
        glm::mat4 m = MakeMoonMatrix( );
        glm::vec4  eyePos = glm::vec4( 0., 0., 0., 1. );  // a position
        glm::vec4 lookPos = glm::vec4( 0., 0., 0., 1. );  // a position
        glm::vec4   upVec  = glm::vec4( 0., 0., 0., 0. ); // a vector, so doesn't get translations applied
        float znear = 0.1f;

        glMatrixMode( GL_MODELVIEW );
        glLoadIdentity( );
        switch( NowView )
        {
            . . .

            case OUTSIDEVIEW:           // 1st way to set gluLookAt( )
                gluLookAt( 0., 0., 30000.f,     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 Viewing

Put this in the Display( ) function:

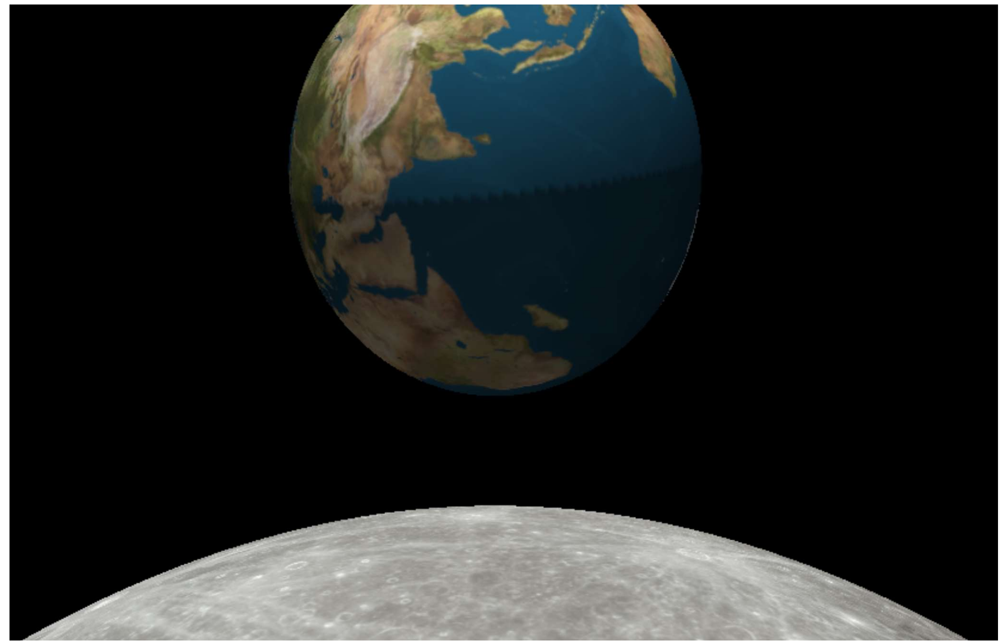
```
void
Display( )
{               . . .
                glm::mat4 e  = MakeEarthMatrix( );
                glm::mat4 m = MakeMoonMatrix( );
                glm::vec4  eyePos = glm::vec4( 0., 0., 0., 1. );
                glm::vec4 lookPos = glm::vec4( 0., 0., 0., 1. );
                glm::vec4   upVec = glm::vec4( 0., 0., 0., 0. );  // vectors don't get translations
                float znear = 0.1f;

                glMatrixMode( GL_MODELVIEW );
                glLoadIdentity( );
                switch( NowView )
                {
                       . . .
                       case EARTHVIEW:                              // 2nd way to set gluLookAt( )
                               SetViewingFromLatLng( 0.f, 70.f,   0.f, 60.f,    EARTH_RADIUS_MILES, &eyePos, &lookPos, &upVec, &znear );
                               eyePos  = e * eyePos;
                               lookPos = e * lookPos;
                               upVec    = e * upVec;
                               gluLookAt( eyePos.x, eyePos.y, eyePos.z,   lookPos.x, lookPos.y, lookPos.z,   upVec.x, upVec.y, upVec.z );
                               break;
        . . .
                }
```

Moon Viewing

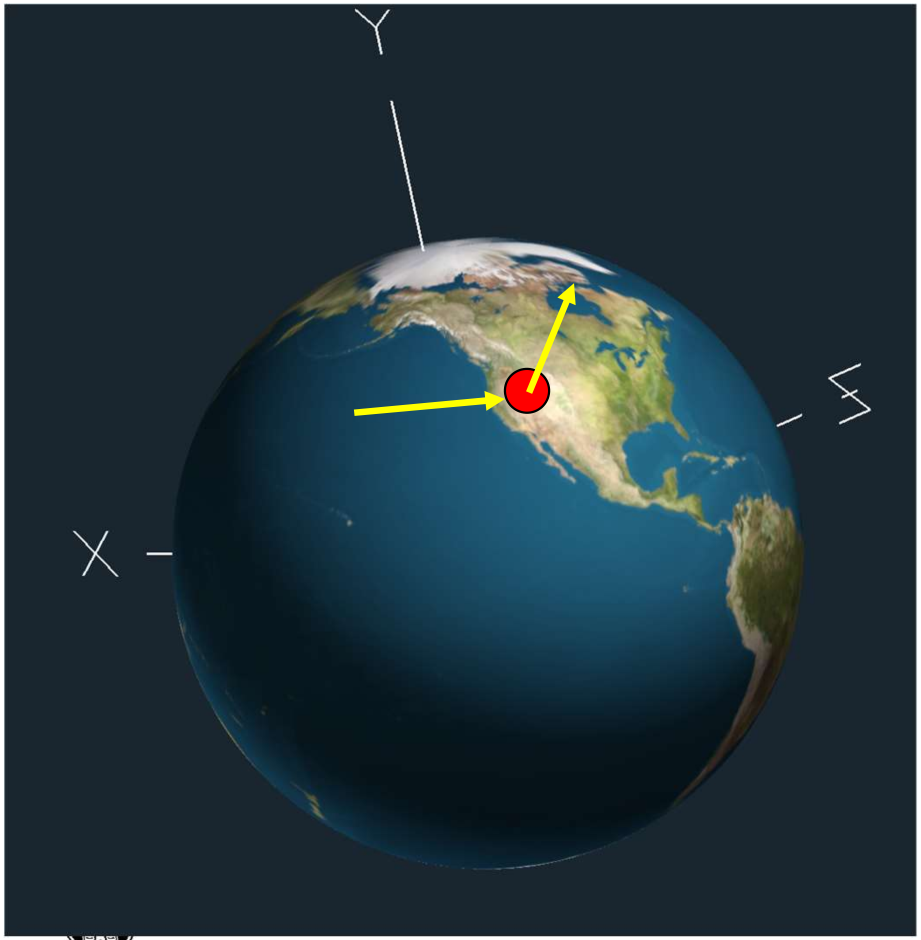
Put this in the Display( ) function:

```
void
Display( )
{               . . .
                glm::mat4 e  = MakeEarthMatrix( );
                glm::mat4 m = MakeMoonMatrix( );
                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
                float znear = 0.1f;

                glMatrixMode( GL_MODELVIEW );
                glLoadIdentity( );
                switch (NowView)
                {
                     . . .
                     case MOONVIEW:                         // 3rd way to set gluLookAt( )
                           SetViewingFromLatLng(0.f, 175.f,   0.f, 170.f,   MOON_RADIUS_MILES, &eyePos, &lookPos, &upVec, &znear);
                           eyePos  = m * eyePos;
                           lookPos = m * lookPos;
                           upVec    = m * upVec;
                           gluLookAt( eyePos.x, eyePos.y, eyePos.z,   lookPos.x, lookPos.y, lookPos.z,   upVec.x, upVec.y, upVec.z );
                           break;            . . .
                }
```

Transformations In Action!



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:

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

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

Because of the way the coordinates work, Corvallis's west longitude needs to positive, even though on maps, west longitude is negative.

Earth/Corvallis Viewing

```
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° );
```

Put this in the Display( ) function:

```
void
Display( )
{
        . . .
        glm::mat4 e = MakeEarthMatrix( );
        glm::mat4 m = MakeMoonMatrix( );
        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
        float znear = 0.1f;

        glMatrixMode( GL_MODELVIEW );
        glLoadIdentity( );
        switch( NowView )
        {
            . . .
            case CORVALLISVIEW:            // 4th way to set gluLookAt( )
                SetViewingFromLatLng(44.57f, 133.27f,   44.57f, 83.27f,   EARTH_RADIUS_MILES, &eyePos, &lookPos, &upVec, &znear);
                eyePos =  e * eyePos;
                lookPos = e * lookPos;
                upVec =    e * upVec;
                gluLookAt( eyePos.x, eyePos.y, eyePos.z,   lookPos.x, lookPos.y, lookPos.z,   upVec.x, upVec.y, upVec.z );
                break;          . . .
        }
```

Transformations In Action!