Possibly Adjusting the Viewing Volume

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

Remember these lines from our sample code?

```c
float znear = 0.01f;
float zfar = 1000000.0f;
```

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

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.

Here's How I Tried Scaling Things

To get your objects to be bigger than the sun, you might have to change the `znear` and `zfar` values. You'd also have to change the `gluPerspective()` values in your call to

```c
gluPerspective(90.0f, 1.0f, znear, zfar);
```

To get your objects to be 2000 miles in diameter, you might have to use

```c
const float EORS = 1.f / 93000000.f;
```

Here are the actual numbers for our solar system; you would need to change them to your exaggerated numbers.

In our short planet example, you might need to change the `znear` and `zfar` values in your call to `gluPerspective()` to work with whatever scale you choose.

Here are the main and clipping plane distances in front of the eye. The ZFAR 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 the factor times the radius of the specific planet.
### Earth Transformations

Steps to transform the Earth-eye-viewing system into Solar System Coordinates:

1. Spin the Earth by EarthSpinAngle about its X axis.
2. Tilt the Earth by 23.5° about its Y axis.
3. Translate the Earth to where it belongs in the Sun (gCallList EarthList).
4. Translate the Moon to where it belongs in the Earth (gCallList MoonList).
5. Revolve the Moon about its Y axis.
6. Revolve the Moon by MoonOrbitAngle about the Earth's Y axis.
7. Translate the Moon to where it belongs in the Earth (gCallList MoonList).

#### Note:

- You can use these matrices to draw the objects in the proper locations instead of using gRotate() and gTranslate().

#### Display List Setup

A display list can call a previously-created display list.

#### Earth Viewing

Steps to transform the Earth-eye-viewing system into Solar System Coordinates:

1. Spin the Earth by EarthSpinAngle about its X axis.
2. Tilt the Earth by 23.5° about its Y axis.
3. Translate the Earth to where it belongs in the Sun (gCallList EarthList).

#### Note:

- You can use these matrices to draw the objects in the proper locations instead of using gRotate() and gTranslate().

#### Transformations in Action!

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

---

### Transformations In Action!

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

---

### Moon Transformations

Steps to transform the Moon-eye-viewing system:

1. Spin the Moon by MoonSpinAngle about its Y axis.
2. Translate the Moon by MoonOrbitPosition in its X direction.
3. Revolve the Moon by MoonOrbitAngle about the Earth's Y axis.
4. Translate the Moon by MoonOrbitPosition in its X direction.
5. Revolve the Moon by MoonOrbitAngle about the Earth's Y axis.
6. Translate the Moon by MoonOrbitPosition in its X direction.

#### Note:

- The orbit angle also does the spin.
- Note that EarthOrbitAngle and EarthTiltAngle have no effect on the Moon's matrix.

---

### Transformations in Action!

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

```cpp
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 eye position and look-at position (in degrees) and a planet radius to an eye position and a look-at position in XYZ coordinates.

```cpp
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 (in degrees) and a planet radius to an (x,y,z) coordinate.

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…)

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.

It would be nice if this was all there was to it. Hint: there’s more!

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!

With the eye and look combination, we are looking up through the planet.

We are about to use some vector math!

Vector math is a big deal in computer graphics. Your games use it all the time.

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

Unit Vector: \[ \hat{V} = \frac{V}{\|V\|} \]

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

\[ A \cdot \hat{B} = \|A\| \|\hat{B}\| \cos \theta \]

\[ A \cdot \hat{B} = \|A\| \hat{B} \]

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

So, how much of A lives in the 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

- **Look-at Position**
- **Eye Position**

Want to move the eye position to this tangent plane component:

\[ \text{perpComponent} = (\text{eyeToLook} \cdot \text{upvec}) \cdot \text{upVec} \]

\[ \text{tangentComponent} = \text{eyeToLook} - \text{perpComponent} \]

The up-vector is a unit vector perpendicular to the planet at the look-at position.

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

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.

This is not just bad, it is very bad!
Here's the Viewing Strategy: Use All These Parameters to Compute $N$:

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 $\text{EYEDISTFACTOR}$.
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 $\text{EYEDISTFACTOR}$.

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 $\text{EYEDISTFACTOR}$.
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 $\text{EYEDISTFACTOR}$.

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 $\text{EYEDISTFACTOR}$.
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 $\text{EYEDISTFACTOR}$.

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 $\text{EYEDISTFACTOR}$.
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 $\text{EYEDISTFACTOR}$.
11/2/2023

Put this in the Display() function:

Moon Viewing

```cpp
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;
        . . .
    }
}
```

Put this in the Display() function:

Earth/Corvallis Viewing

```cpp
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;
        . . .
    }
}
```

What if you want the eye at Corvallis (or some other arbitrary location)?

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

- \( y = \sin(44.57°) \approx 0.702 \)
- \( x = \cos(44.57°) \approx 0.712 \)
- \( z = -x \cos(123.27°) \approx -0.391 \)
- \( x = EARTH_RADIUS_MILES \times x \cos(123.27°) \approx 15.53 \)
- \( z = EARTH_RADIUS_MILES \times x \sin(123.27°) \approx 15.53 \)

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.