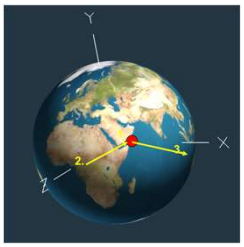
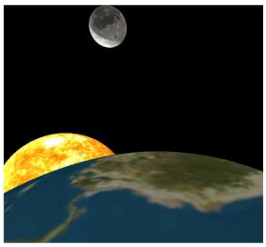
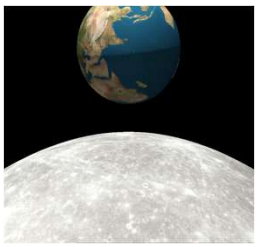
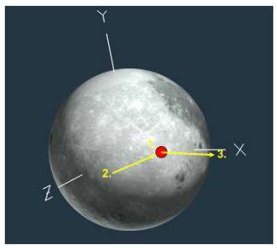
1

Placing the Eye Position on an Orbiting Body

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

Oregon State University
Computer Graphics

2

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

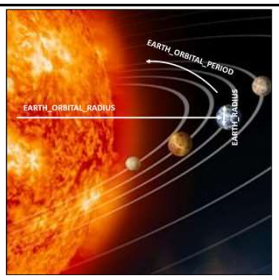
Which eye view is the current one

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

Oregon State University
Computer Graphics

mjb – November 2, 2023

Possibly Adjusting the Viewing Volume



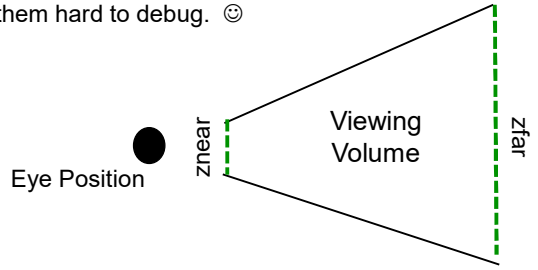
Remember these lines from our sample code?

```
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*.

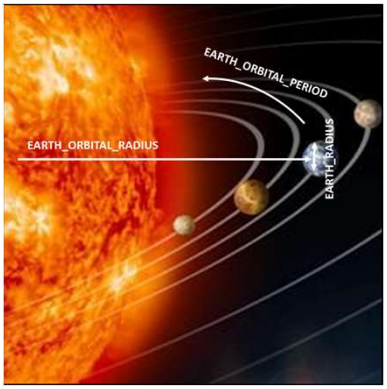
This makes them hard to debug. ☺



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:



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

```
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;
```

***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.

Oregon State University
Computer Graphics

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;
```

Oregon State University Computer Graph

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 );
. . .
```

Oregon State University Computer Graphics

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

Oregon State University
Computer Graphics

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

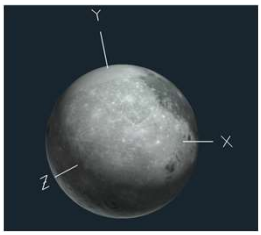
} $[M_{e/s}]$

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  [ M_e/s ]
}
```

University
Computer Graphics

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

} $[M_{m/e}]$

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

} $[M_{e/s}]$

glCallList(MoonList);

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}
```

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

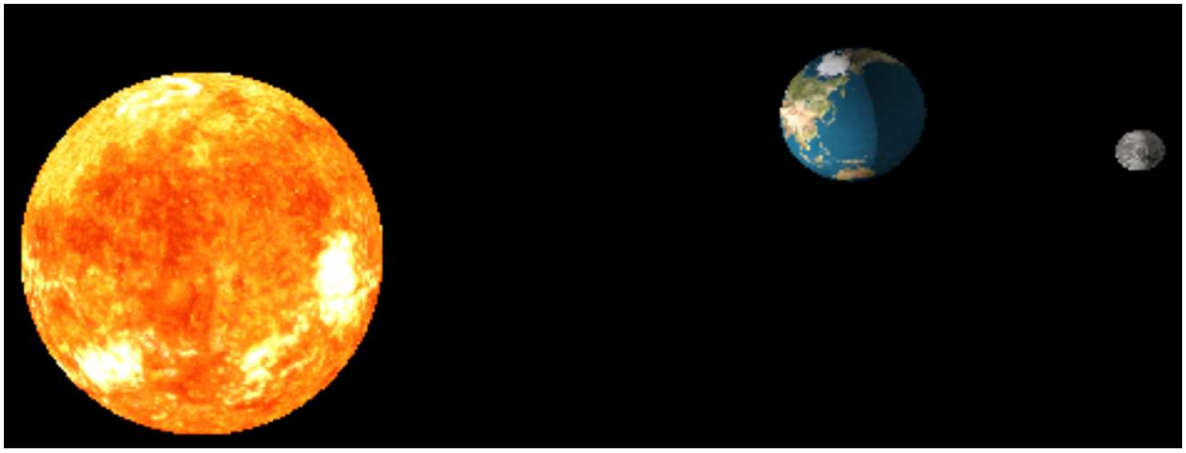
Note: You Can Use these Matrices to Draw the Objects in the Proper Locations instead of using glRotatef( ) and glTranslatef( )


```
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( );
```

Oregon State University
Computer Graphics

Transformations In Action!

11

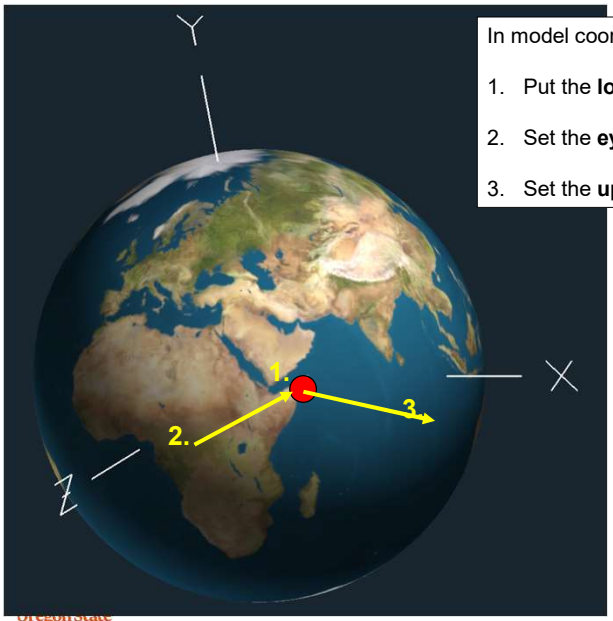


Oregon State University
Computer Graphics

mjb – November 2, 2023

Earth Viewing

12



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…).

Oregon State University
Computer Graphics

mjb – November 2, 2023

Converting Latitude-Longitude to XYZ

13

```
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.

Oregon State University
Computer Graphics

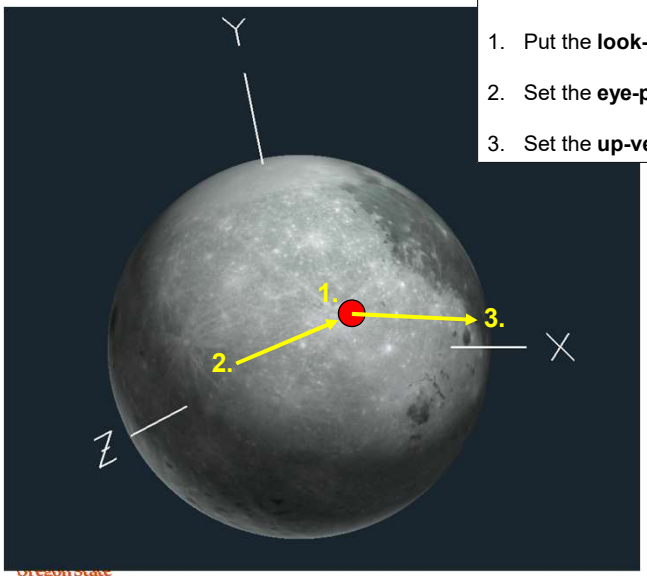
mjb – November 2, 2023

Moon Viewing

14

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…).

Oregon State University
Computer Graphics

mjb – November 2, 2023

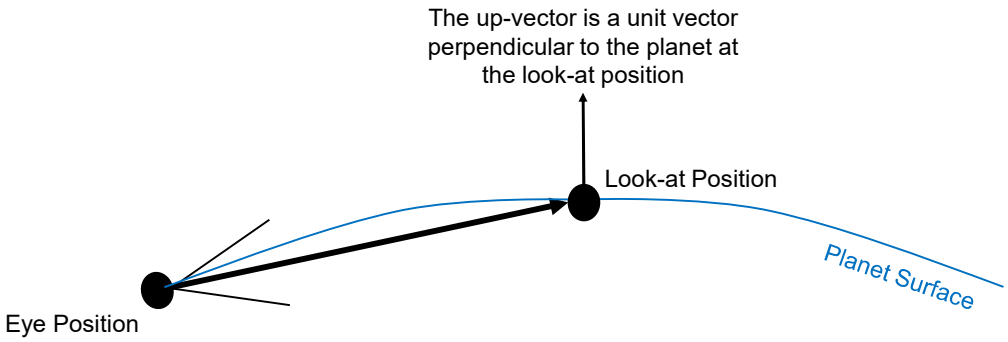
Converting Eye+Look Latitude-Longitude to Eye+Look XYZ: It would be nice if this was all there was to it. Hint: there's more!


```
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.

Oregon State University
Computer Graphics

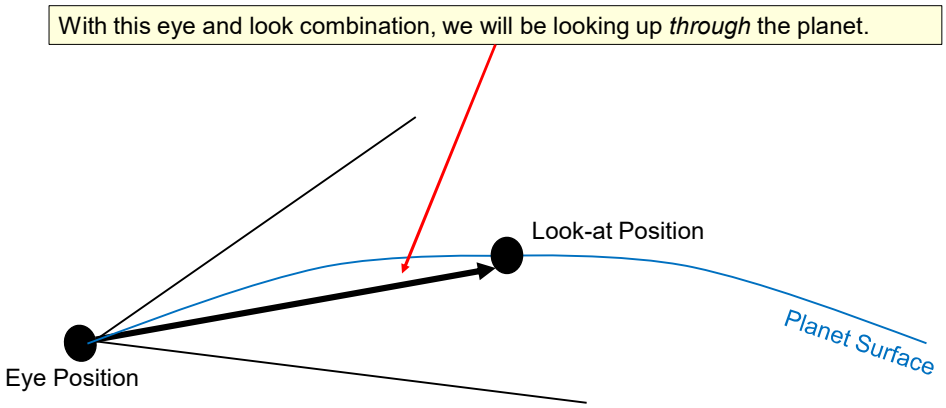
Here's the Viewing Strategy



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

Oregon State University
Computer Graphics

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



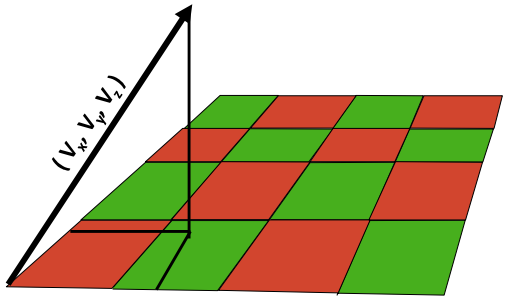
We are about to use some vector math!
Vector math is a *big deal* in computer graphics. Your games use it all the time.
You don't have to completely understand vector math to appreciate what we are about to use it for.

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

19



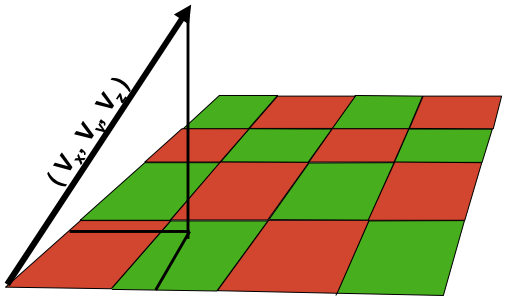
Magnitude: $\|V\| = \sqrt{V_x^2 + V_y^2 + V_z^2}$

Oregon State University
Computer Graphics

mjb – November 2, 2023

Sidebar: Unit Vectors have a Magnitude = 1.0

20



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

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

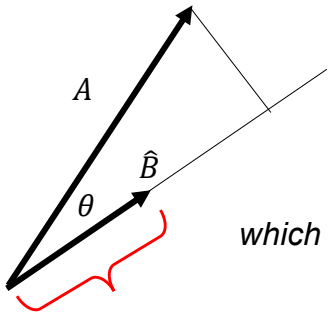
The circumflex (^) tells us this is a unit vector

Oregon State University
Computer Graphics

mjb – November 2, 2023

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

21



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

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

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

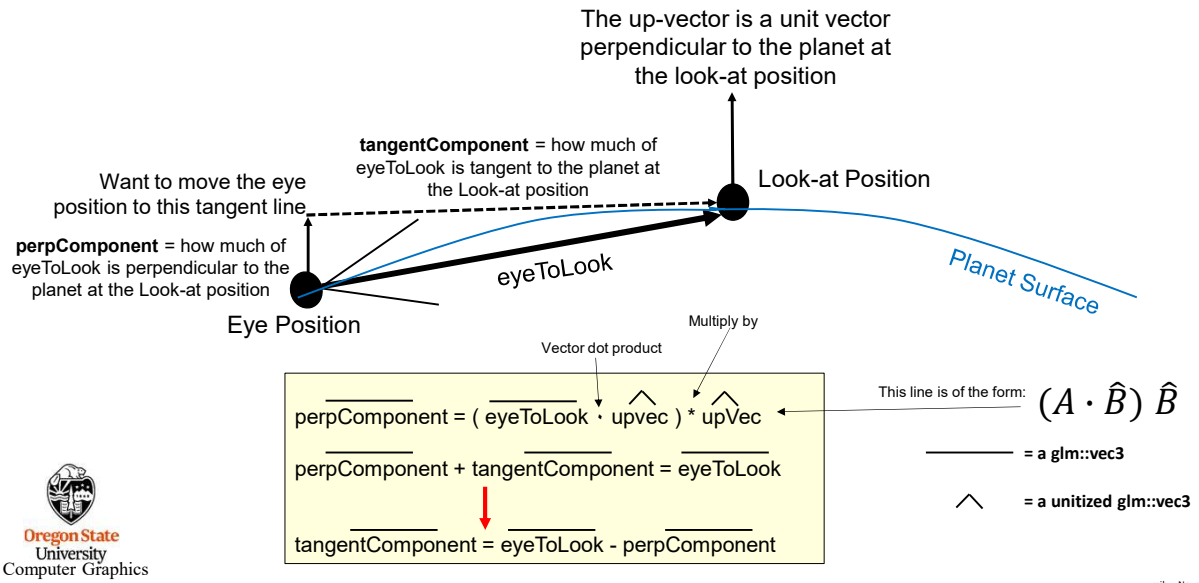
So, how much of A lives in the $\hat{B}$ direction is that magnitude times the B unit vector:

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

Oregon State University Computer Graphics

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

22

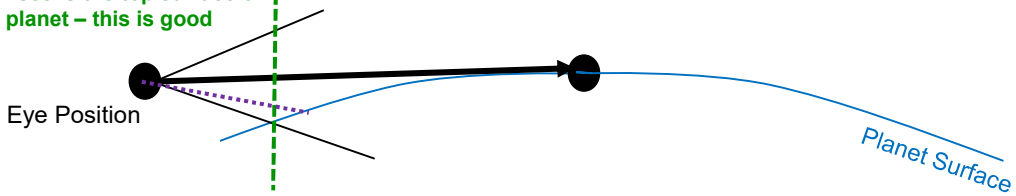


Oregon State University Computer Graphics

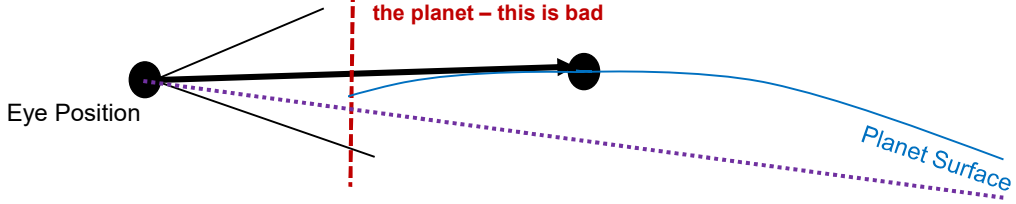
But There is *Another* Problem – We Have to Set the Near Clipping Plane Carefully

23

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



Oregon State University Computer Graphics

mjb – November 2, 2023

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

24

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

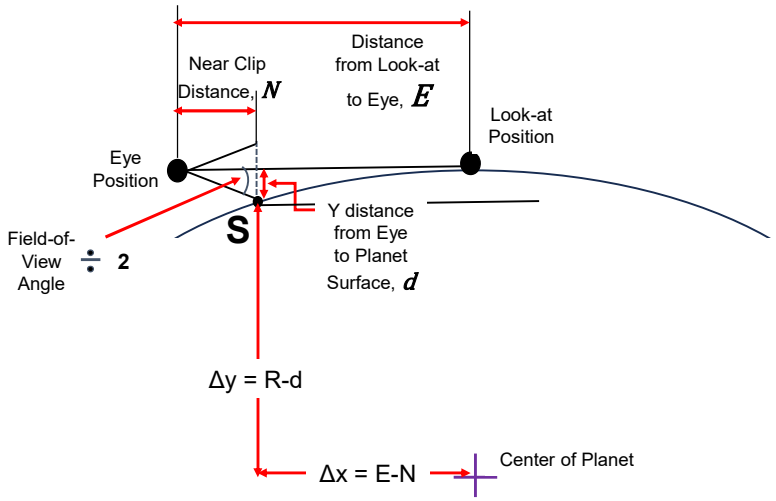
We need to place the eye and the eye's near clipping plane such that no part of what the eye can see is under the planet's surface.

Oregon State University Computer Graphics

mjb – November 2, 2023

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

25



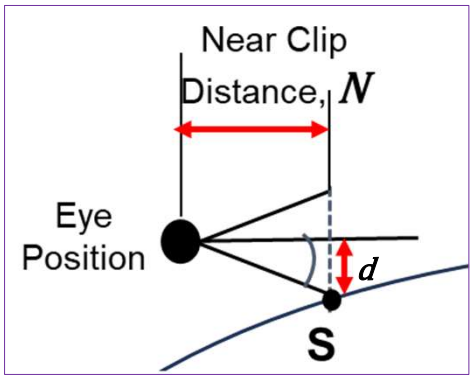
Oregon State University Computer Graphics

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

26



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

$$d = NT$$

At the point S:

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

$$\downarrow$$

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

$$\downarrow$$

$$R^2 - 2RNT + N^2T^2 + (fR)^2 - 2fRN + N^2 = R^2$$

$$\downarrow$$

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

$$\downarrow$$

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

$$\downarrow$$

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

Oregon State University Computer Graphics

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**.

Oregon State University
Computer Graphics

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

Oregon State University Computer Grap

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;
    . . .
        }
```

Oregon State University Computer Graphics

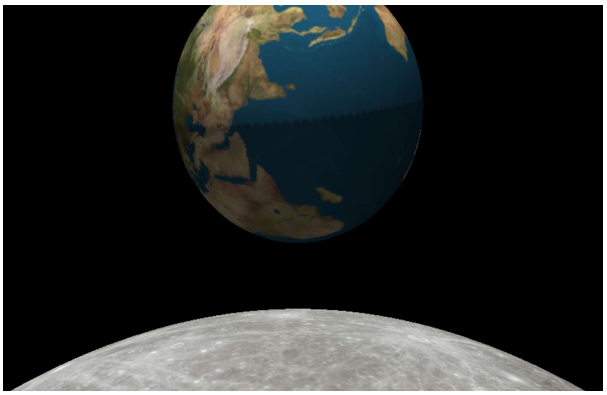
Moon Viewing

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;              . . .
            }
```

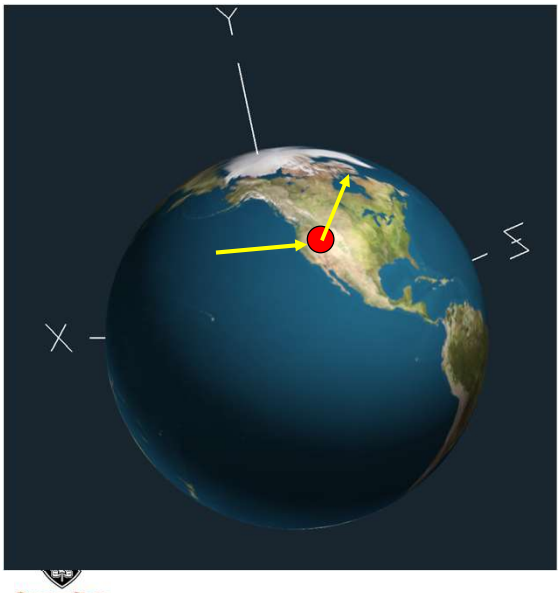
Oregon State University
Computer Graphics

Transformations In Action!



Oregon State University
Computer Graphics

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.

Oregon State University
Computer Graphics

Earth/Corvallis Viewing

Put this in the Display( ) function:

```
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° );
```

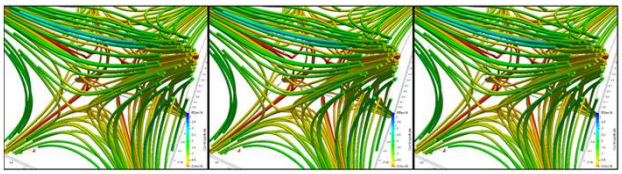
```
void
Display( )
{
        . . .
        glm::mat4 e = MakeEarthMatrix( );
        glm::mat4 m = MakeMoonMatrix( );
        glm::vec4  eyePos = glm::vec4( 0., 0., 0., 1. );
        glm::vec4 lookPos = glm::vec4( 0., 0., 0., 1. );
        glm::vec4   upVec  = glm::vec4( 0., 0., 0., 0. );          // 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;                  . . .
        }
```

Oregon State University
Computer Graphics

Transformations In Action!

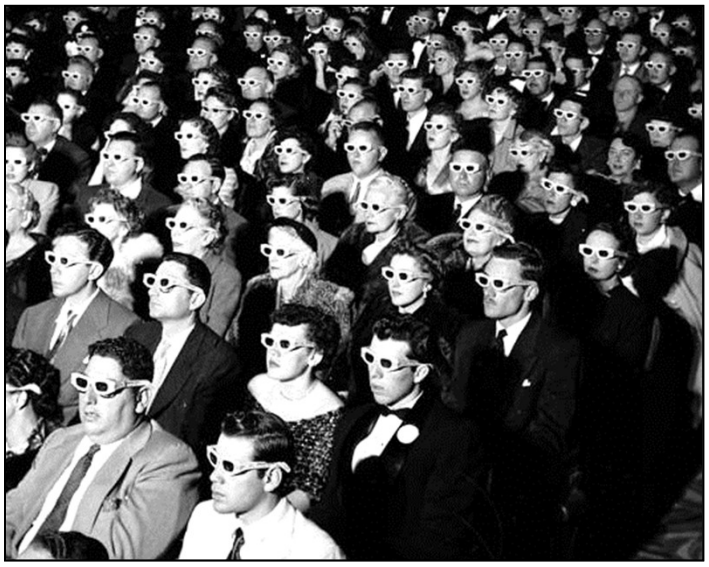
Stereographics

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



Oregon State University
Computer Graphics

Stereovision is not new – It's been in common use in the movies since the 1950s



Life Magazine

Hard to believe that people used to dress like this to go see a movie…

Oregon State University
Computer Graphics

And, even longer than that in stills

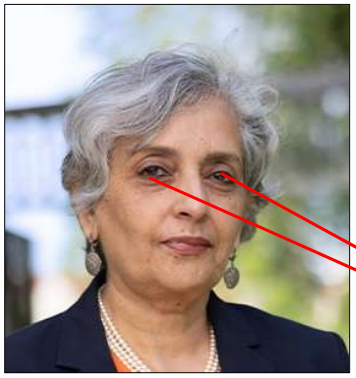
Newport Maritime Museum

Portland Art Museum's Ansel Adams Exhibit

Binocular Vision

In everyday living, part of our perception of depth comes from the slight difference in how our two eyes see the world around us. This is known as *binocular vision*.

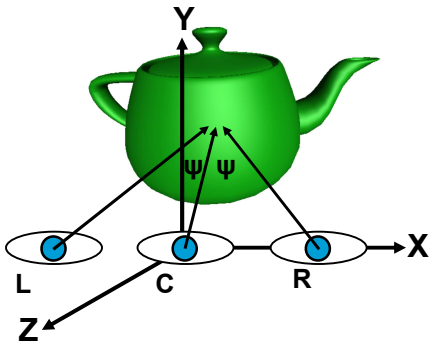
We care about this, and are discussing it, because stereo computer graphics can be a great help in de-cluttering a complex 3D scene. It can also enhance the feeling of being immersed in a movie.



OSU's 16th President Dr. Jayathi Murthy

The Cyclops Model

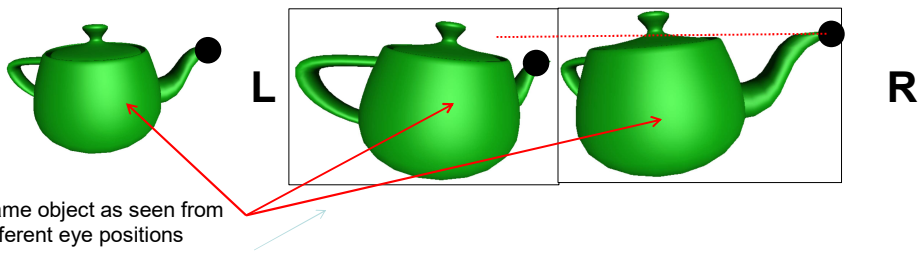
In the world of computer graphics, the two eye views can be reconstructed using standard projection mathematics. The simplest approach is the *Cyclops Model*. In this model, the left and right eye views are obtained by rotating the scene plus and minus what a Cyclops at the origin would see



The left eye view is obtained by rotating the scene an angle +ψ about the Y axis. The right eye view is obtained by rotating the scene an angle -ψ about the Y axis. In practice, if you wanted to do this (and you don't), a good value of ψ would be 1-4˚.

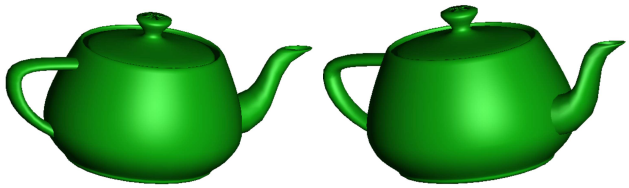
The Vertical Parallax Problem

This seems too simple, and in fact, it is. This works OK if you are doing orthographic projections, but if you use perspective, you will achieve a nasty phenomenon called ***vertical parallax***, as illustrated below:

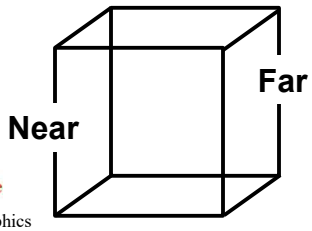


The fact that the perspective shortening causes the black-dot point to have different vertical positions in the left and right eye views makes it very difficult for the eyes to converge the two images. For perspective projections, we need a better way.

The Vertical Parallax Problem



Why not just keep using orthographic projections? Mathematically this is fine, but in practice, the two depth cues, stereo and no-perspective, fight each other. This will bring on an optical illusion. A good example of this is a simple cube, drawn below using an orthographic projection:



Because of the use of stereographics, the *binocular cues* will say that the Near face is closer to the viewer than the Far face is.

However, our *visual experience* reminds us that the only way a far object can appear the same size as a near object is if it is, in fact, larger. Thus, your visual system will perceive the Far face as being larger than the Near face, when in fact they are the same size.

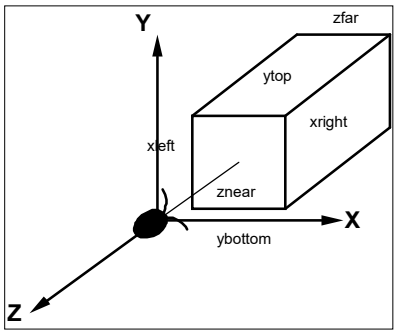
Diversion #1 – Specifying the Viewing Frustum

The OpenGL **glFrustum** call can be used to ask for a perspective projection in place of the **gluPerspective** call:

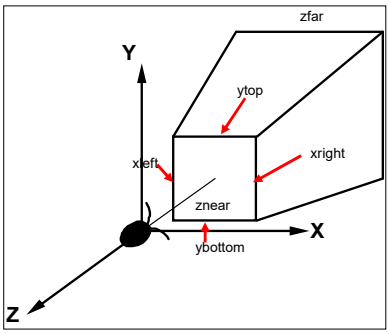
glFrustum(left, right, bottom, top, near, far);

This is meant to look a lot like the **glOrtho( )** call.

In the glFrustum case, the values of **left**, **right**, **bottom**, and **top** are now the boundaries of the viewing volume on the **face of the near clipping plane**. **near** and **far** are the same as used in glOrtho and gluPerspective.



glOrtho(xl, xr, yb, yt, zn, zf);



glFrustum(xl, xr, yb, yt, zn, zf);

Diversion #1 – Specifying the Viewing Frustum

glFrustum(left, right, bottom, top, near, far);

But rather than having to specify the left, right, bottom, and top limits at the face of the near clipping plane (which is what **glFrustum** expects), let's setup a way to specify those limits at some convenient distance in front of us. (This is derived using similar triangles.)

```
void
FrustumZ( float left, float right, float bottom, float top, float znear,   float zfar,   float zproj )
{
        if( zproj != 0.0 )
        {
                left    *= ( znear/zproj );
                right   *= ( znear/zproj );
                bottom  *= ( znear/zproj );
                top     *= ( znear/zproj );
        }

        glFrustum( left, right, bottom, top, znear, zfar );
}
```

A convenient distance in front of the eye to set the left, right, bottom, and top values

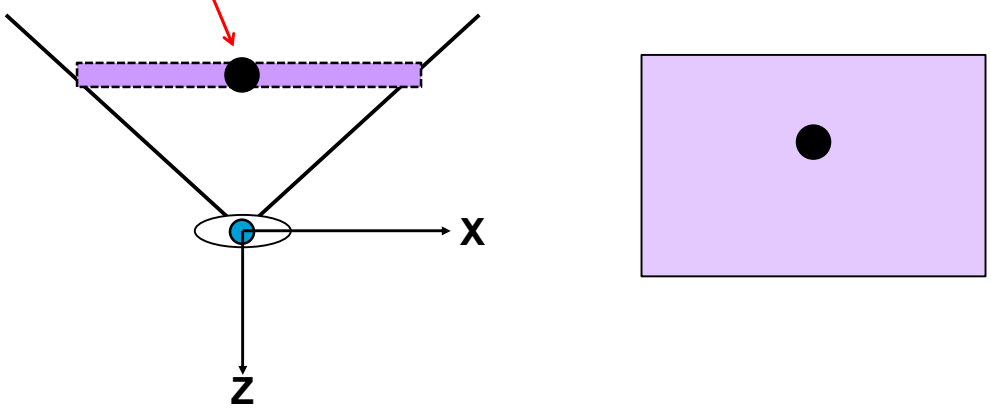
Oregon State University Computer Graphics

So, if you wanted to view a car from 30 feet away, you could say:

FrustumZ(-10., 10., -10., 10., .1, 100., 30.);

Diversion #2 – Where does a 3D Point Map to in a 2D Window?

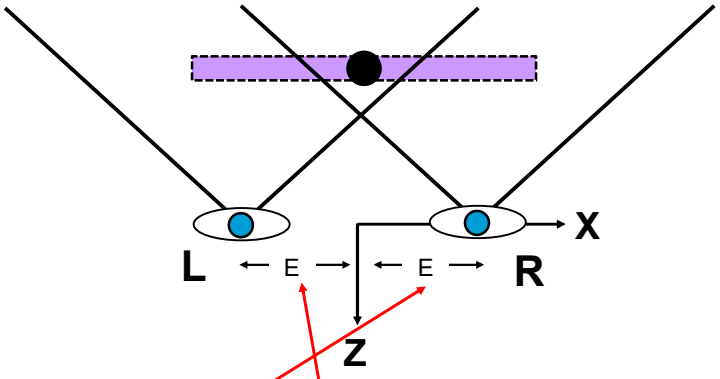
Take an arbitrary 3D point in the viewing volume. Place a plane parallel to the near and far clipping planes at its Z value (i.e., depth in the frustum). The location of the point on that plane shows proportionally where the 3D point will be perspective-mapped from left to right in the 2D window.



Oregon State University Computer Graphics

Two Side-by-side Perspective Viewing Volumes

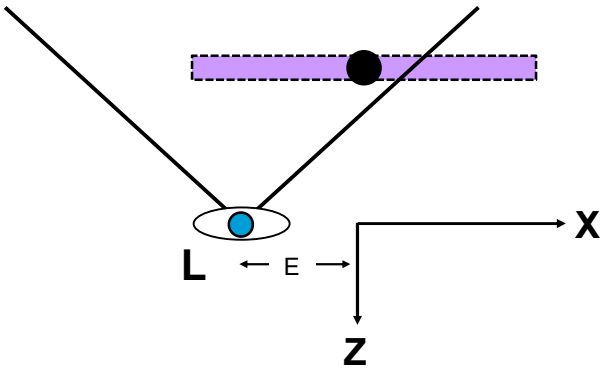
The best stereographics work is done with perspective projections. To avoid the vertical parallax problem, we keep both the left and right eyes looking straight ahead so that, in the vertical parallax example shown before, the block-dot point will project with exactly the same amount of perspective shortening.



The left eye view is obtained by translating the eye by -E in the X direction, which is actually accomplished by translating the scene by +E instead. Similarly, the right eye view is obtained by translating the scene by -E in the X direction.

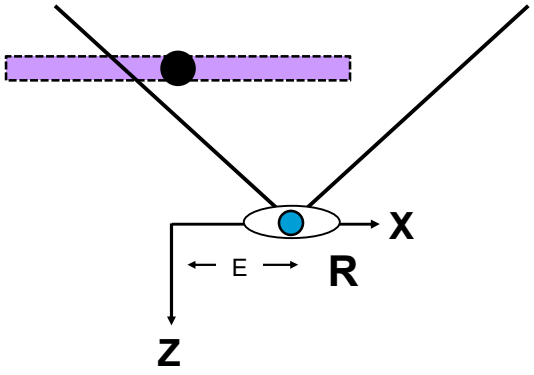
Two Side-by-side Perspective Viewing Volumes

The best stereographics work is done with perspective projections. To avoid the vertical parallax problem, we keep both the left and right eyes looking straight ahead so that, in the vertical parallax example shown before, the block-dot point will project with exactly the same amount of perspective shortening.



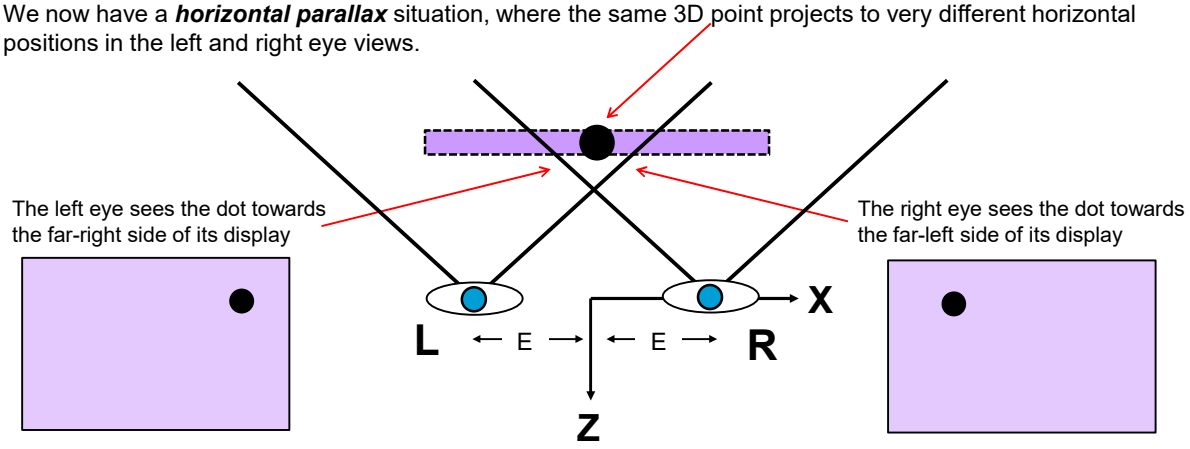
Two Side-by-side Perspective Viewing Volumes

The best stereographics work is done with perspective projections. To avoid the vertical parallax problem, we keep both the left and right eyes looking straight ahead so that, in the vertical parallax example shown before, the block-dot point will project with exactly the same amount of perspective shortening.



Two Side-by-side Perspective Viewing Volumes

We now have a ***horizontal parallax*** situation, where the same 3D point projects to very different horizontal positions in the left and right eye views.

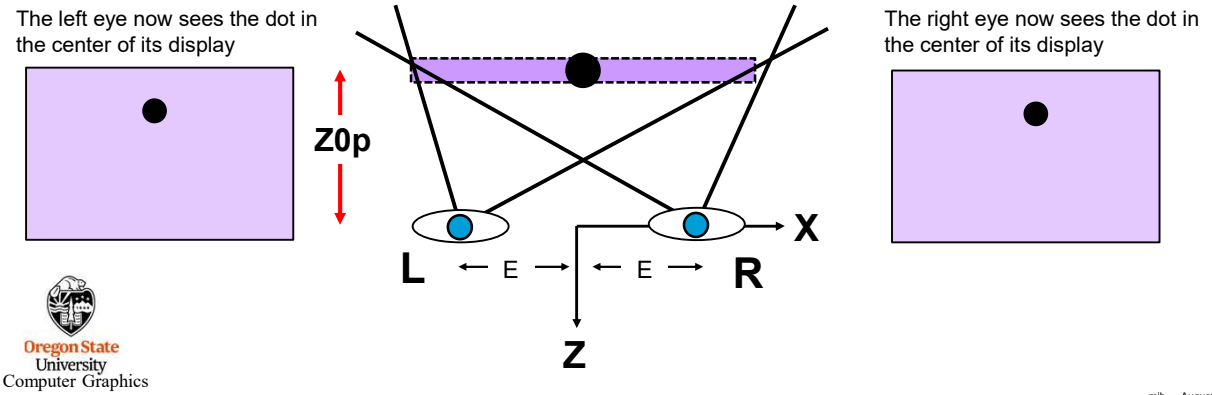


Note that this is a *situation*, not a *problem*. The difference in the left and right eye views requires at least *some* horizontal parallax in order to work for stereographics. You can convince yourself of this by alternately opening and closing your left and right eyes. We just need a good way to *control* the horizontal parallax to keep it convergeable by your eyes.

Two Side-by-side Non-symmetric Perspective Viewing Volumes

We do this by moving the sides of each eye's viewing volume to match the left and right boundaries of the cyclops-eye's viewing volume. We also define a distance in front of the eye, ***z0p***, to the ***plane of zero parallax***. This is where our 3D dot now projects to the ***same location for each eye's display.***

To the viewer, the plane of zero parallax will be the glass monitor screen and objects in front of it will appear to live in the air in front of the glass screen and objects behind this plane will appear to live inside the monitor.



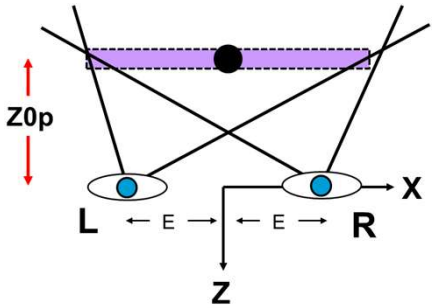
Oregon State University
Computer Graphics

Two Side-by-side *Non-symmetric* Perspective Viewing Volumes

Use the Cyclops's left and right boundaries as the left and right boundaries for each eye, even though the scene has been translated. In the left eye view, the boundaries must then be shifted by +E to match the +E shift in the scene. In the right eye view, the boundaries must be shifted by -E to match the -E shift in the scene.

Looking from the Cyclops eye at the origin, determine the left, right, bottom, and top boundaries of the viewing window on the plane of zero parallax as would be used in a call to glFrustum( ). These can be determined by knowing Z0p and the original field-of-view angle Φ:

Left eye viewing volume:
*L0p = -Z0p * tan(ø/2) + E*
*R0p = Z0p * tan(ø/2) + E*



Right eye viewing volume:
*L0p = -Z0p * tan(ø/2) - E*
*R0p = Z0p * tan(ø/2) - E*

Oregon State University
Computer Graphics

```
void
Stereopersp( float fovy, float aspect, float znear, float zfar, float z0p, float eye )
{
        float left, right;      // x boundaries on z0p
        float bottom, top;      // y boundaries on z0p
        float tanfovy;          // tangent of y fov angle

        // tangent of the y field-of-view angle:

        tanfovy = tan( fovy * (M_PI / 180.) / 2. );

        // top and bottom boundaries:

        top = z0p * tanfovy;
        bottom = -top;

        // left and right boundaries come from the aspect ratio:

        right = aspect * top;
        left =  aspect * bottom;

        // take eye translation into account:

        left  -= eye;
        right -= eye;

        // ask for a window in terms of the z0p plane:

        FrustumZ( left, right, bottom, top, znear, zfar, z0p );

        // translate the scene opposite the eye translation:

        glTranslatef( -eye, 0.0, 0.0 );

}
```


An Example

Parallel viewing stereo

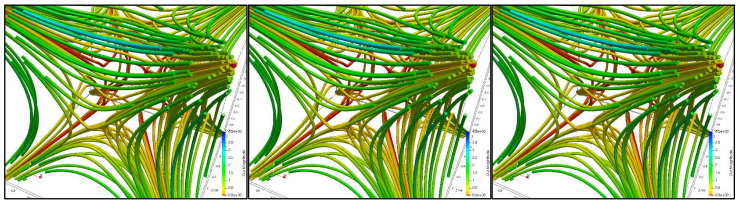
Left Right

Cross-eye viewing stereo

Right Left

Oftentimes, Stereographics Images are printed like this so that both Parallel and Cross-eyed Viewing will Work

L **R** **L**



Print this page and cut out the left two images

Oregon State University
Computer Graphics

Note to self: don't resize these images, as much as you are tempted to – they fit perfectly in ypur viewer as they are now.

Acquiring Stereo Photos Yourself: A Two-camera Mounting Bar

Places to mount this bar to a tripod

Places to mount two cameras

Oregon State University
Computer Graphics

Acquiring Stereo Photos Yourself: A Digital Stereo Camera

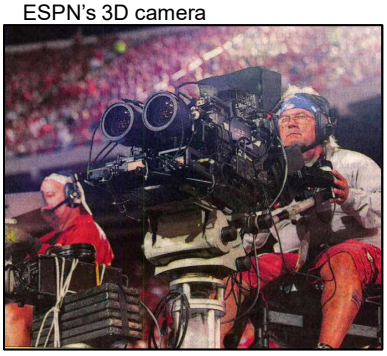
Two lenses

Acquiring Stereo Video

ESPN's 3D camera

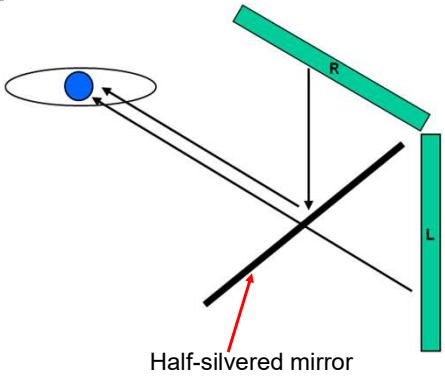
Panasonic's 3D Camcorder

ESPN's 3D camera



Separating the Left and Right-eye Views: The View-Master

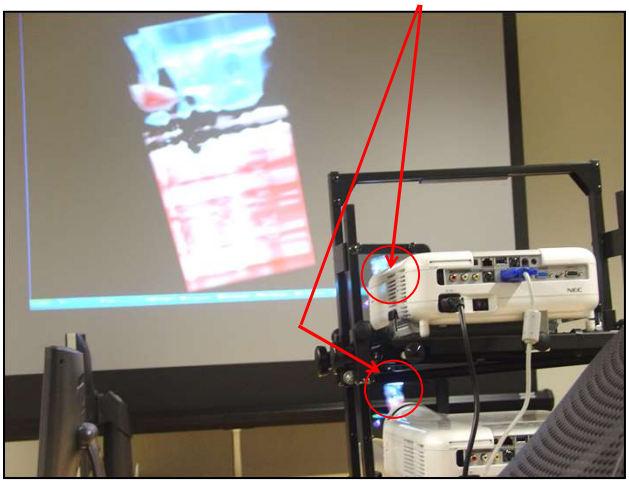
Separating the Left and Right-eye Views: Stereo Mirror



An interesting property of mirrors is that the reflection rotates the polarization by 90º so that the two images can be separated by polarized glasses.

Separating the Left and Right-eye Views: Dual Projectors ("GeoWall")

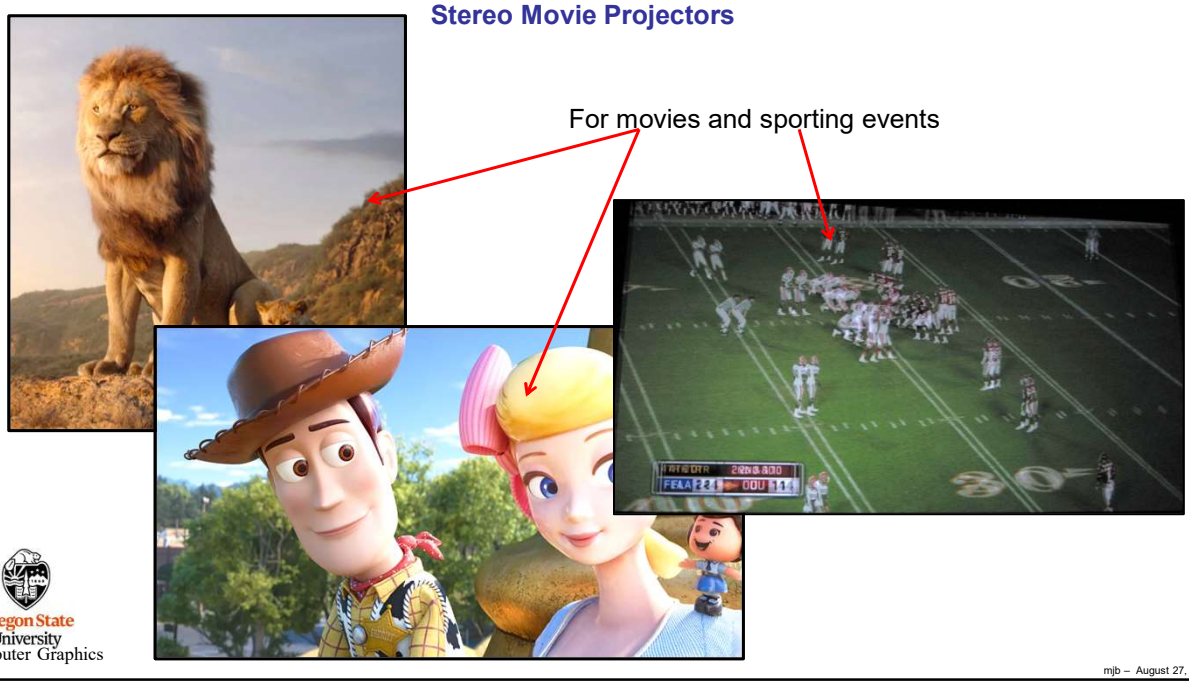
Two filters statically provide the polarization



Oregon State University
Computer Graphics

Separating the Left and Right-eye Views: Stereo Movie Projectors

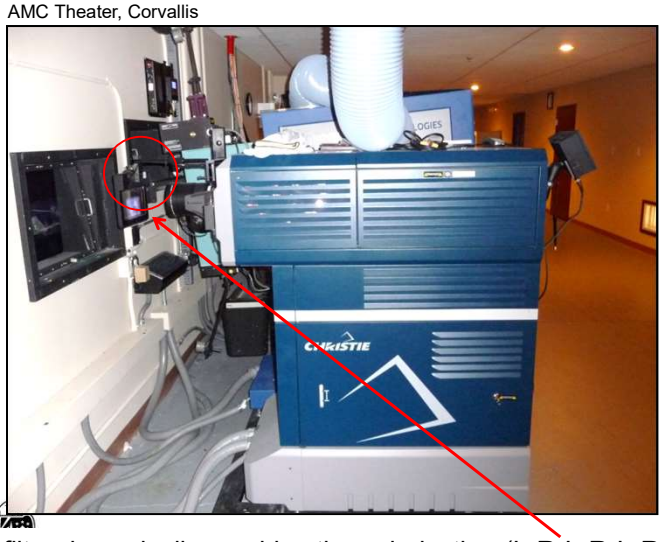
For movies and sporting events



Oregon State University
Computer Graphics

Separating the Left and Right-eye Views: Stereo Movie Projectors

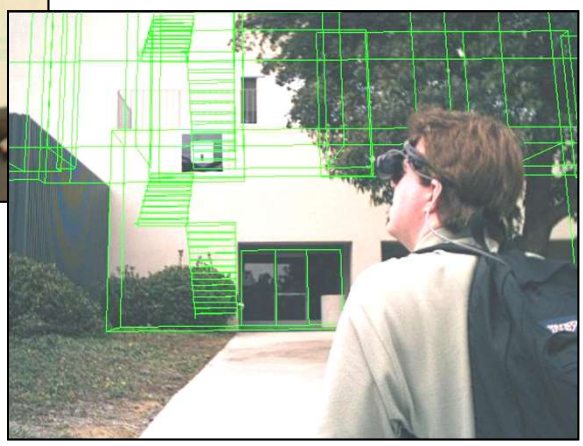
AMC Theater, Corvallis



Circularly polarized glasses

One filter dynamically provides the polarization (L-R-L-R-L-R per $^1/_{24}$ sec frame).
These are the projectors and glasses that the Corvallis AMC movie theater uses.

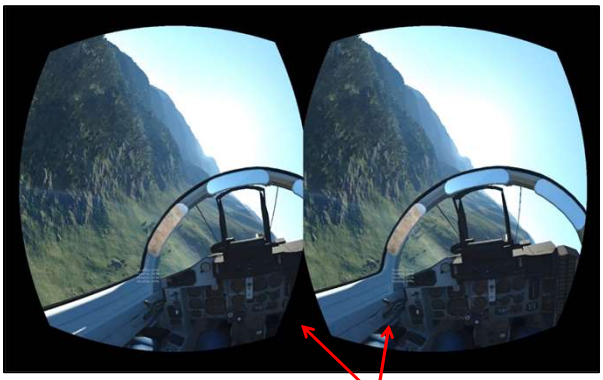
Separating the Left and Right-eye Views: Head-mounted Goggles



Separating the Left and Right-eye Views: VR Headsets

Uses an accelerometer and a gyroscope to determine the head position and the head orientation

http://theriftarcade.com



Uses shaders to get the correct fisheye lens distortion

Oregon State University Computer Graphics

Separating the Left and Right-eye Views – View-Master Viewer for your Cell phone

Uses the phone's gyroscope to know the head orientation

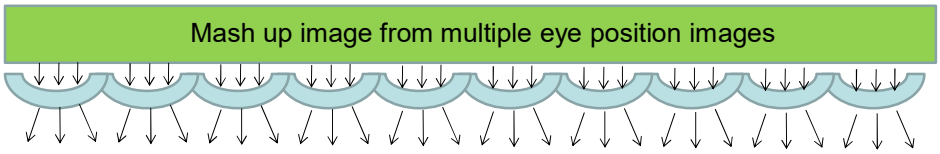
SideKick

Uses a moving magnet and the phone's digital compass to perform a "left-click"

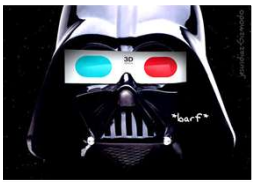
Uses shaders to get the correct fisheye lens distortion

Oregon State University Computer Graphics

Separating the Left and Right-eye Views: Lenticular

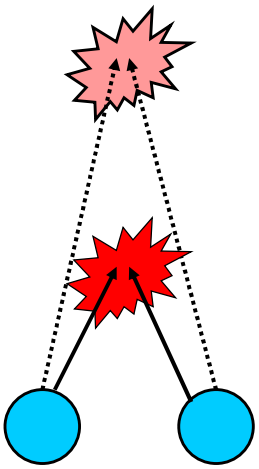


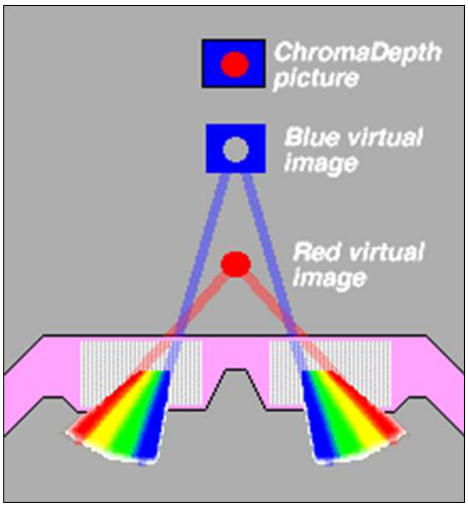
Stereographics Rules of Thumb



- Stereographics is especially good for de-cluttering wireframe displays.
- Use perspective, not orthographic, projections to avoid the optical illusion.
- Use an eye separation, E, of approximately: $E = Z0p * \tan(1^\circ\text{-}4^\circ)$
- Use the far clipping plane well. The stereo effects are enhanced when the scene is not complicated by a lot of tiny detail that is far away. The interactive response is improved too.
- Because you are drawing the scene twice, using display lists is especially important.
- It is fun to set Z0p = Zfar so that the image appears to be hanging out in the air in front of the monitor. However, in real life we rarely see anything hanging out in the air that has its sides clipped for no apparent reason, as your scene is likely to have. Perceptually, it is often better to set Z0p = Znear so that the entire scene looks like it is inside the monitor and that you are viewing it through a rectangular hole cut through the glass. This situation is common in everyday life, so we are used to seeing things that way. Personally, I like putting Z0p about 1/3 of the way through the scene.
- Intensity depth cueing (glFog) nicely enhances the stereo illusion. This also lets you bring the far clipping plane in closer.
- If you are using texture mapping, be sure to use GL_LINEAR, not GL_NEAREST, for the texture filtering.

Encoding Stereo in a Single Image – ChromaDepth™

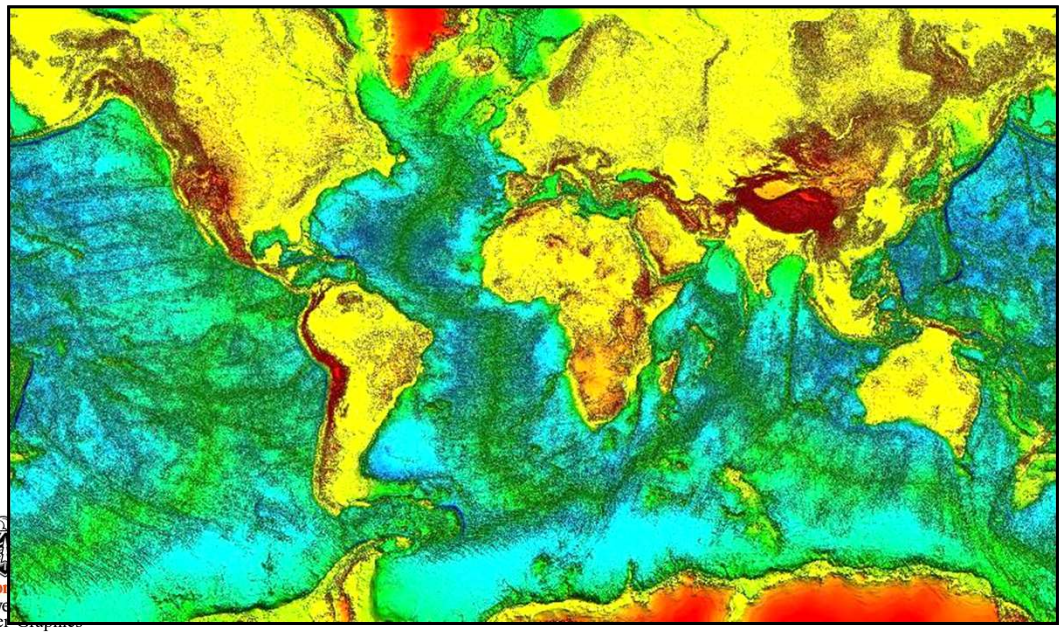




Oregon State University
Computer Graphics

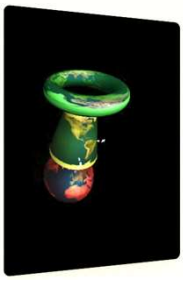
http://cs.oregonstate.edu/~mjb/chromadepth

Encoding Stereo in a Single Image – ChromaDepth™



Oregon State University
Computer Graphics

Looking Glass Quilts for a Web Page- Embedded 3D Display, Using both OpenGL and Blender



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

Oregon State University
Computer Graphics

Here is How to Generate a Quilt from OpenGL

Here are some pre-defined constants you will need:

```
const int QUILTWIDTH  = 3360;
const int QUILTHEIGHT = 3360;
const int QUILTNUMWIDTH  = 8;
const int QUILTNUMHEIGHT = 6;
const int QUILTTOTALBLOCKS = QUILTNUMWIDTH * QUILTNUMHEIGHT;
const int BLOCKWIDTH  = QUILTWIDTH  / QUILTNUMWIDTH;                    // 420
const int BLOCKHEIGHT = QUILTHEIGHT / QUILTNUMHEIGHT;                   // 560
const float QUILTZ0P = 5.f;
const float QUILTDELTAEYE = 0.10f;
const float QUILT_ASPECT_Y_OVER_X = (float)BLOCKHEIGHT / (float)BLOCKWIDTH; // 4:3 = 1.3333

const char * QUILTFILENAME = "Quilt_qs8x6a0.75.bmp";        // must be in the format "*_qs8x6a0.75.*"

unsigned char QuiltArray[QUILTHEIGHT][QUILTWIDTH][3];                   // holds the entire quilt
unsigned char BlockArray[BLOCKHEIGHT][BLOCKWIDTH][3];                   // holds one block
```

Oregon State University
Computer Graphics

Triggering the File Output


```
void
Keyboard( unsigned char c, int x, int y )
{
	switch( c )
	{
		case 'w':
		case 'W':
			CreateAndWriteQuilt( );
			break;

		. . .

		case 'q':
		case 'Q':
		case ESCAPE:
			DoMainMenu( QUIT );	// will not return here
			break;				// happy compiler

		default:
			fprintf( stderr, "Don't know what to do with keyboard hit: '%c' (0x%0x)\n", c, c );
	}

	// force a call to Display( ):

	glutSetWindow( MainWindow );
	glutPostRedisplay( );
}
```

Hit the 'w' key to trigger the 48 renderings and file-writing

Oregon State University
Computer Graphics

Generating the 48 Individual Images for the Quilt, I


```
void
CreateAndWriteQuilt( )
{
	glutSetWindow( MainWindow );

	float eyex = - QUILTDELTAEYE*(float)(QUILTTOTALBLOCKS)/2.f;
	for (int y = 0; y < QUILTNUMHEIGHT; y++ )
	{
		for( int x = 0; x < QUILTNUMWIDTH; x++ )
		{
			glDrawBuffer(GL_FRONT);
			glEnable(GL_DEPTH_TEST);
			glShadeModel(GL_FLAT);
			int vx = glutGet(GLUT_WINDOW_WIDTH);
			int vy = glutGet(GLUT_WINDOW_HEIGHT);
			int xl = (vx - BLOCKWIDTH) / 2;
			int yb = (vy - BLOCKHEIGHT) / 2;
			glViewport(xl, yb, BLOCKWIDTH, BLOCKHEIGHT);
			glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

			glMatrixMode( GL_PROJECTION );
			glLoadIdentity( );
			OffAxisPersp( 70.f, QUILT_ASPECT_Y_OVER_X, 0.1f, 1000.f, QUILTZ0P, eyex );

			glMatrixMode(GL_MODELVIEW);
			glLoadIdentity( );
			gluLookAt(0.f, 0.f, 8.f, 0.f, 0.f, 0.f, 0.f, 1.f, 0.f);

			DisplayScene( );

			glFlush();
			glFinish();

			. . .
```

Oregon State University
Computer Graphics

Generating the 48 Individual Images for the Quilt, II


```
                        . . .

                        // done rendering -- upload the pixels:

                        glReadBuffer(GL_FRONT);
                        glPixelStorei(GL_PACK_ALIGNMENT, 1);
                        glReadPixels(xl, yb, BLOCKWIDTH, BLOCKHEIGHT, GL_RGB, GL_UNSIGNED_BYTE, BlockArray);

                        //  location for this block in the large QuiltArray:

                        for( int yb = 0; yb < BLOCKHEIGHT; yb++)
                        {
                                for( int xb = 0; xb < BLOCKWIDTH; xb++)
                                {
                                        QuiltArray[y*BLOCKHEIGHT+yb][x*BLOCKWIDTH+xb][0] = BlockArray[yb][xb][0];
                                        QuiltArray[y*BLOCKHEIGHT+yb][x*BLOCKWIDTH+xb][1] = BlockArray[yb][xb][1];
                                        QuiltArray[y*BLOCKHEIGHT+yb][x*BLOCKWIDTH+xb][2] = BlockArray[yb][xb][2];
                                }
                        }

                        Sleep(250);

                        eyex += QUILTDELTAEYE;
                }  // x
        }  // y

        WriteArray( QUILTFILENAME );
}
```

Oregon State University
Computer Graphics

DisplayScene( ) – just do the scene drawing, I


```
void
DisplayScene()
{
        // rotate the scene:

        glRotatef((GLfloat)Yrot, 0.f, 1.f, 0.f);
        glRotatef((GLfloat)Xrot, 1.f, 0.f, 0.f);

        // uniformly scale the scene:

        if (Scale < MINSCALE)
                Scale = MINSCALE;
        glScalef((GLfloat)Scale, (GLfloat)Scale, (GLfloat)Scale);

        // possibly draw the axes:

        if (AxesOn != 0)
        {
                glColor3fv(&Colors[WhichColor][0]);
                glCallList(AxesList);
        }

        // since we are using glScalef( ), be sure the normals get unitized:

        glEnable(GL_NORMALIZE);
        . . .
```

Oregon State University
Computer Graphics

DisplayScene( ) – just do the scene drawing, II


```
        . . .

        glShadeModel(GL_FLAT);
        if (SmoothOn)
        {
                glShadeModel(GL_SMOOTH);
        }

        glDisable(GL_LIGHTING);
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
        if (LightingOn)
        {
                glEnable(GL_LIGHTING);
                glEnable(GL_LIGHT0);
                glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
        }

        glBindTexture(GL_TEXTURE_2D, Tex);
        glDisable(GL_TEXTURE_2D);
        if (TextureOn)
        {
                glEnable(GL_TEXTURE_2D);
        }

        . . .
```

Oregon State University
Computer Graphics

DisplayScene( ) – just do the scene drawing, III


```
        . . .

        // draw the objects by calling up their display lists:

        glTranslatef(0.f, -2.25f, 0.);
        glColor3f(0.8f, 0.2f, 0.2f);
        SetMaterial(0.8f, 0.2f, 0.2f, 10.f);
        glCallList(SphereDL);

        glTranslatef(0.f, 1.75f, 0.);
        glColor3f(0.8f, 0.8f, 0.2f);
        SetMaterial(0.8f, 0.8f, 0.2f, 8.f);
        glCallList(ConeDL);

        glTranslatef(0.f, 2.5f, 0.);
        glColor3f(0.2f, 0.8f, 0.2f);
        SetMaterial(0.2f, 0.8f, 0.2f, 6.f);
        glCallList(TorusDL);

        glDisable(GL_LIGHTING);
        glDisable(GL_TEXTURE_2D);
        glShadeModel(GL_FLAT);
}
```

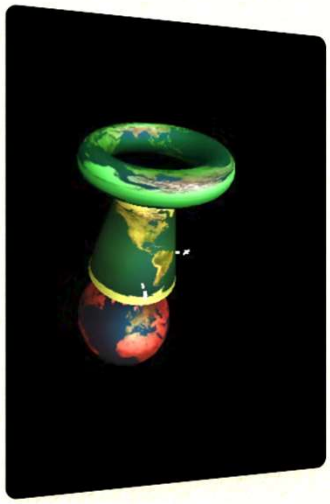
Oregon State University
Computer Graphics

Doing the Off-Axis Projection


```
void
FrustumZ( float left, float right, float bottom, float top, float znear, float zfar,   float zproj )
{
	if( zproj != 0.0 )
	{
		left   *= ( znear/zproj );
		right  *= ( znear/zproj );
		bottom *= ( znear/zproj );
		top    *= ( znear/zproj );
	}

	glFrustum( left, right, bottom, top, znear, zfar );
}

void
OffAxisPersp( float fovxdeg, float aspect_y_over_x, float znear, float zfar, float z0p, float eyex )
{
	float tanfovx = tanf( (float)(M_PI/180.) * fovxdeg / 2.f );

	float right = z0p * tanfovx;
	float left  = -right;

	float bottom = aspect_y_over_x * left;
	float top    = aspect_y_over_x * right;

	left  -= eyex;
	right -= eyex;

	FrustumZ( left, right, bottom, top, znear, zfar, z0p );

	glTranslatef( -eyex, 0.0, 0.0 );
}
```

Oregon State University
Computer Graphics

Quilt_qs8x6a0.75.bmp



Oregon State University
Computer Graphics

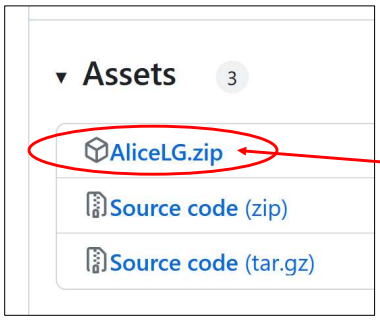
How the usual Display( ) function calls DisplayScene( )

```
void
Display( )
{
	glutSetWindow( MainWindow );
	glDrawBuffer( GL_BACK );
	glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
	glEnable( GL_DEPTH_TEST );
	glShadeModel( GL_FLAT );

	// set the viewport to a square centered in the window:
	GLsizei vx = glutGet( GLUT_WINDOW_WIDTH );
	GLsizei vy = glutGet( GLUT_WINDOW_HEIGHT );
	GLsizei v = vx < vy ? vx : vy;                  // minimum dimension
	GLint xl = ( vx - v ) / 2;
	GLint yb = ( vy - v ) / 2;
	glViewport( xl, yb,  v, v );

	// set the viewing volume:
	glMatrixMode( GL_PROJECTION );
	glLoadIdentity( );
	if( WhichProjection == ORTHO )
		glOrtho( -2.f, 2.f,     -2.f, 2.f,     0.1f, 1000.f );
	else
		gluPerspective( 70.f, 1.f,      0.1f, 1000.f );

	// place the objects into the scene:
	glMatrixMode( GL_MODELVIEW );
	glLoadIdentity( );
	gluLookAt( 0.f, 0.f, 7.f,     0.f, 0.f, 0.f,     0.f, 1.f, 0.f );

	// draw just the scene:
	DisplayScene( );

	glutSwapBuffers( );
	glFlush( );
}
```


Here is How to Generate a Quilt from Blender

You can also generate these quilts automatically from Blender.

The first thing you need to do is retrieve a Blender Add-on.
The Add-on's name is **AliceLG.zip** and it can be found at https://github.com/regcs/AliceLG/releases

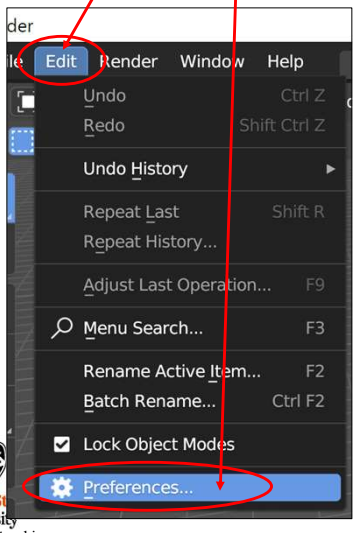


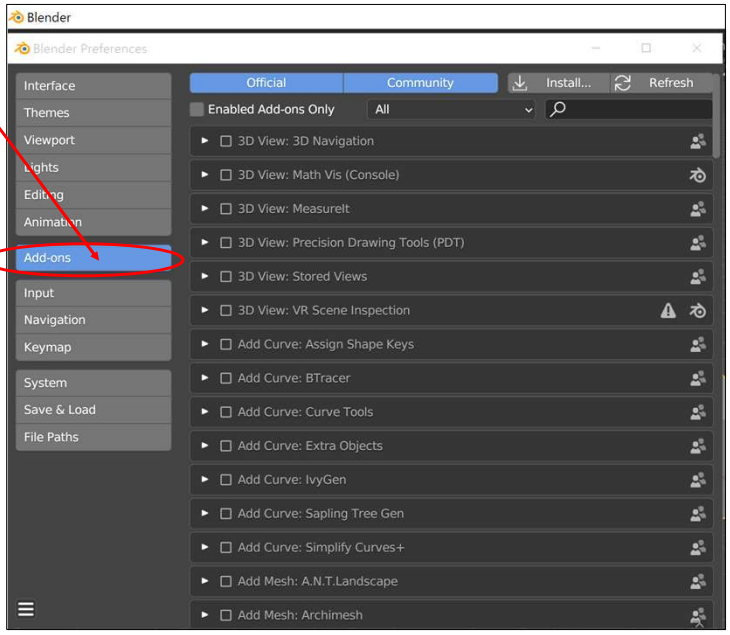
Click here and save this file anywhere.
Do not un-zip it!

Getting the AliceLG.zip Blender Add-on, I

Run Blender.

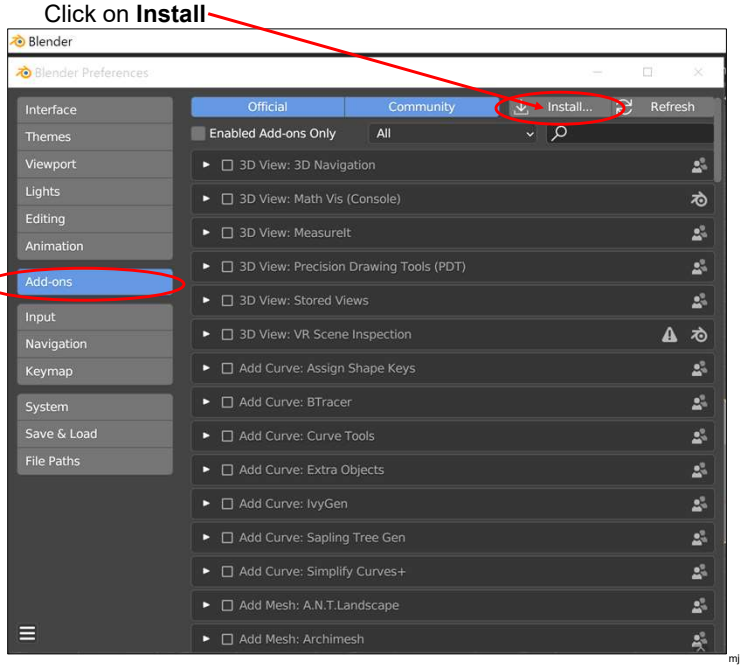
Click on **Edit→Preferences →Add-ons.**





Getting the AliceLG.zip Blender Add-on, II

Click on **Install**

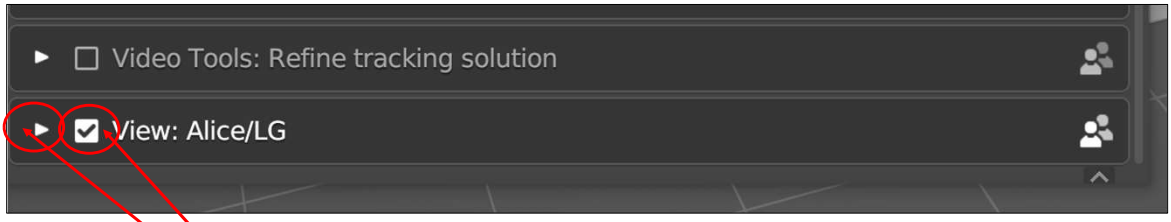


Then navigate to where you installed **AliceLG.zip** and select it.

Don't un-zip it!

Getting the AliceLG.zip Blender Add-on, III

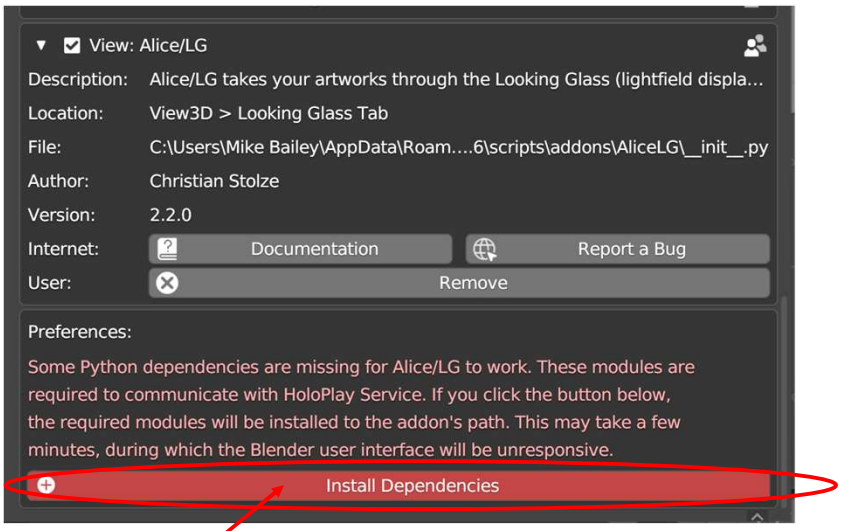
Scroll down the list of Add-ons until you see **View: Alice/LG**



Click the **checkbox** to enable it.
Then click the **arrow** and scroll down so you can see all the information.

Getting the AliceLG.zip Blender Add-on, IV

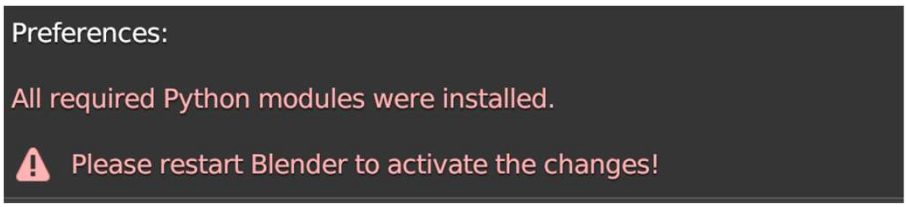
Scroll down so you can see all the information.



Then click on **Install Dependencies**. This might take a while. Be patient.

Getting the AliceLG.zip Blender Add-on, V

When it's done, you will see this:

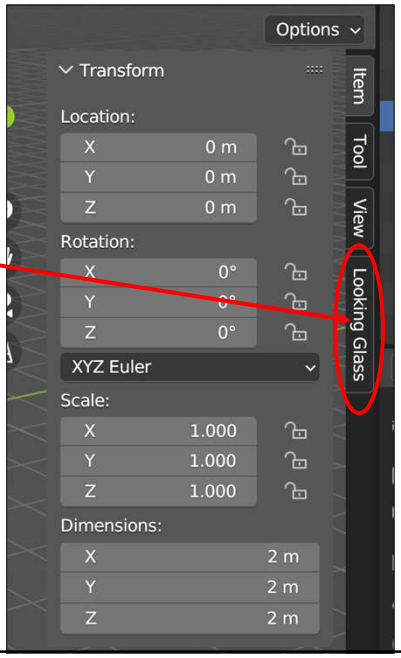


Now exit Blender and start it up again.

Finding the Looking Glass Tab in Blender

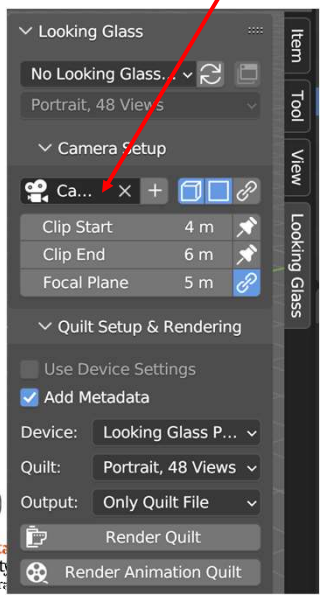
Hit the **'n'** key and you will see the usual Transform panel, but something new has been added.

Click on the **Looking Glass tab**.

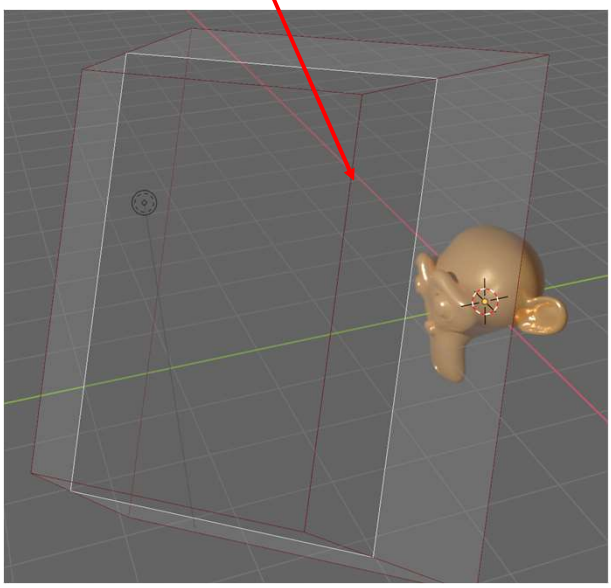


Using the Looking Glass Tab, I

Click here and select **Camera**

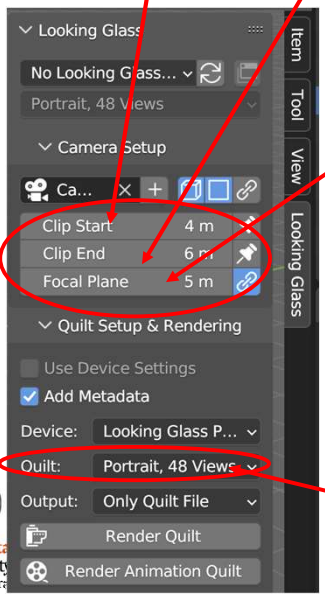


This will bring up a viewing volume, like this:

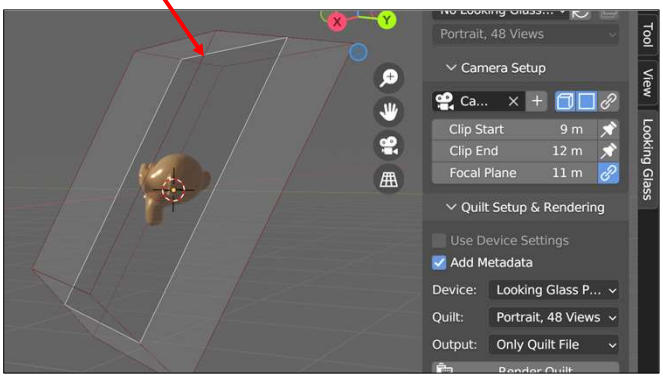


Using the Looking Glass Tab, II

Now adjust the **Clip Start** and **Clip End** to completely surround your scene.



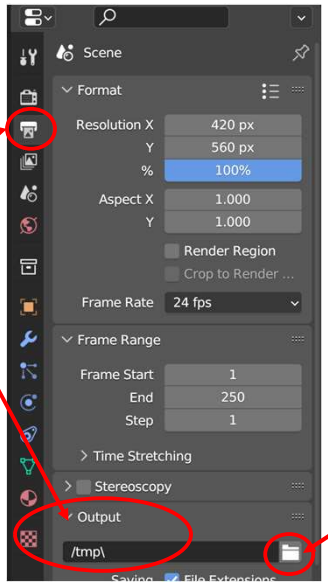
Then adjust the **Focal Plane** to be roughly down the middle of the scene, like this:

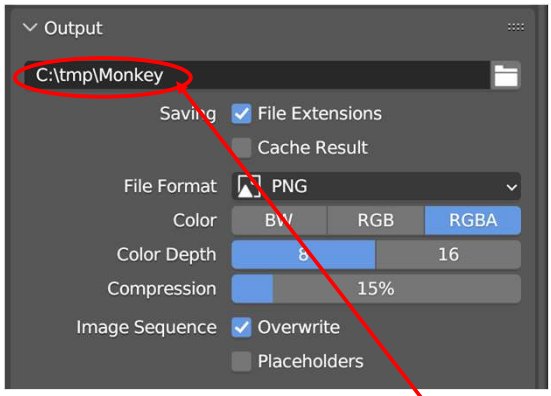


Select **Portrait, 48 Views**

Using the Looking Glass Tab, III

Go select the **Output** icon and navigate down to **Output**.

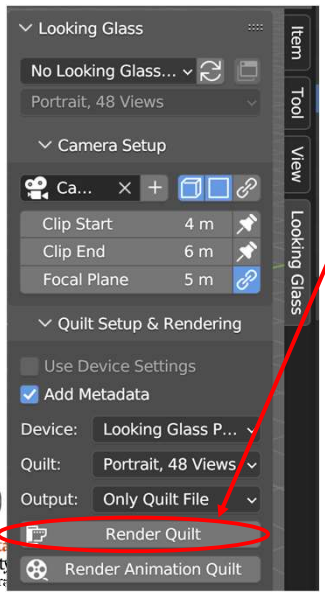




Click here and navigate to some folder for Blender to render your quilt into. Click Accept and then give a filename. On my system, it ended up looking like this.

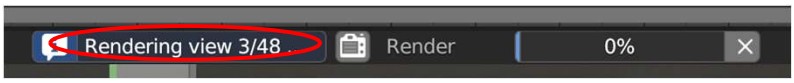
Oregon State University
Computer Graphics

Using the Looking Glass Tab, IV



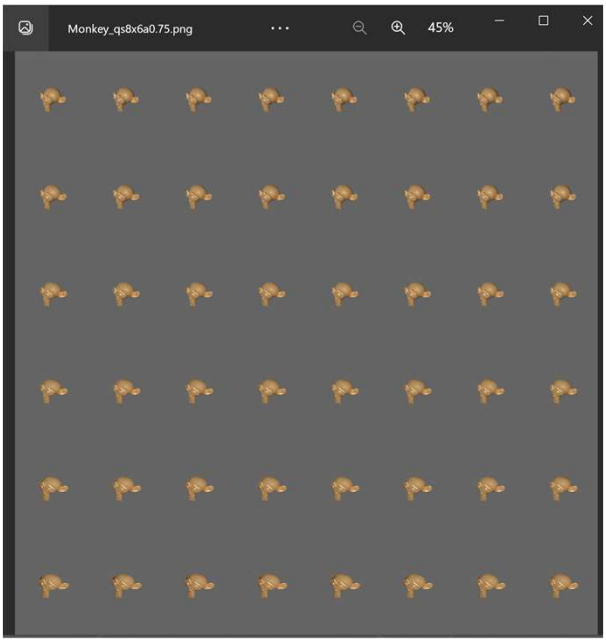
Going back to the Looking Glass tab, click on **Render Quilt**. This will trigger the rendering of 48 different scenes.

You can watch its progress at the bottom of the screen:



Oregon State University
Computer Gra

Monkey_qs8x6a0.75.png

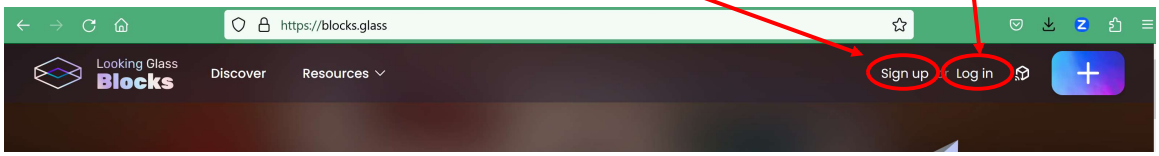


How to Go From a Quilt to a Live "Hologram", I

Navigate to https://blocks.glass

If you don't have an account, **sign up** to get one. If you are under 18 years old, be sure to have your parents help you do this!

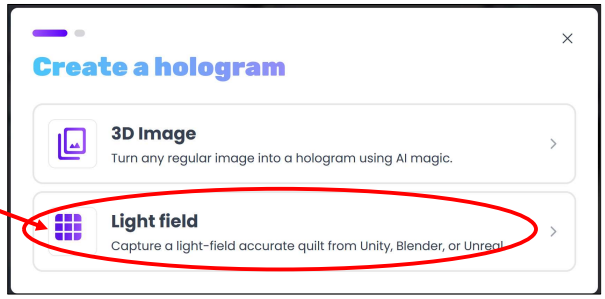
If you do have an account, **Log in** to it.



How to Go From a Quilt to a Live "Hologram", II

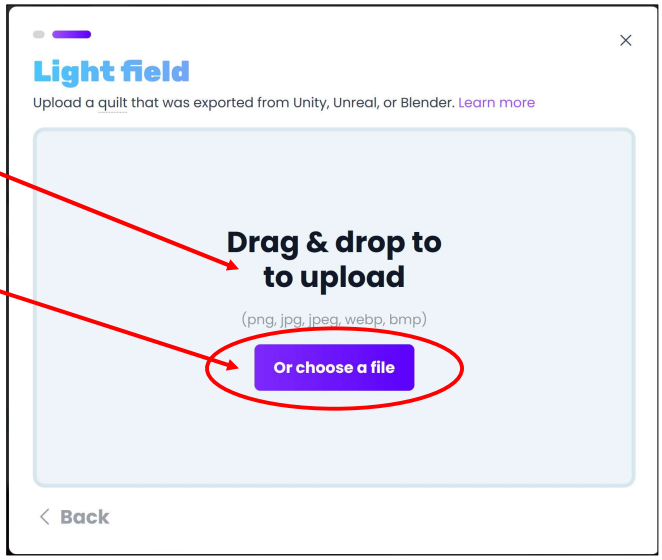
Click on the big Plus Sign

To download a Blender or OpenGL quilt, click here.



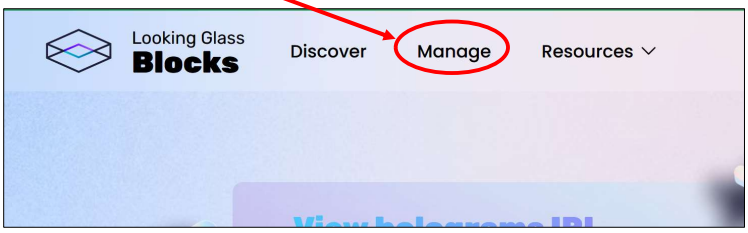
How to Go From a Quilt to a Live "Hologram", III

Drag-and-Drop your quilt file here, or click here and navigate



How to Go From a Quilt to a Live "Hologram", IV

Click on **Manage**, then click on the name of the file you just uploaded



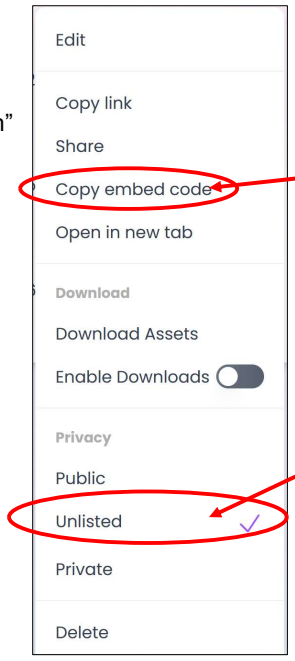
Wiggle the mouse left and right to see your hologram move in 3D

How to Go From a Quilt to a Live "Hologram", V

Click on **Manage** again then click here

How to Go From a Quilt to a Live "Hologram", VI

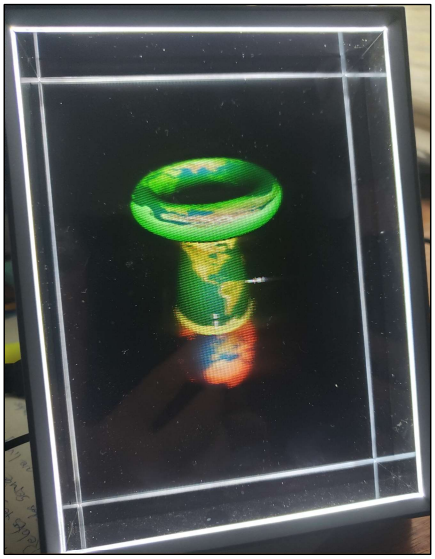
That will bring up a bunch of things you can do with your new "hologram"



Clicking here will copy the embedded HTML code, to get at this hologram from the web, to the clipboard. Paste that into one of your HTML web pages!

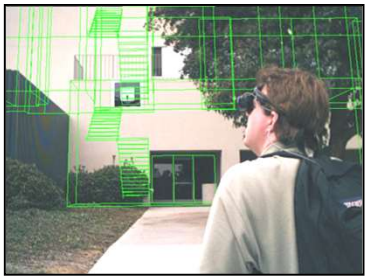
Be sure this permission is set to **Unlisted**.

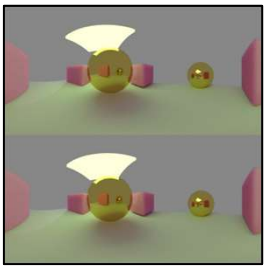
"Casting" a Quilt to a 3D Image on a Looking Glass Portrait Display



https://lookingglassfactory.com/looking-glass-portrait

Virtual and Augmented Reality





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

"Reality, what a concept!"
-- Robin Williams

Virtual Reality Definition

Virtual reality (**VR**) is a simulated experience that can be similar to or completely different from the real world. Applications of virtual reality can include entertainment (i.e. video games) and educational purposes (i.e. medical or military training). Other, distinct types of VR style technology include augmented reality and mixed reality, sometimes referred to as extended reality or XR.

Currently standard virtual reality systems use either virtual reality headsets or multi-projected environments to generate realistic images, sounds and other sensations that simulate a user's physical presence in a virtual environment. A person using virtual reality equipment is able to look around the artificial world, move around in it, and interact with virtual features or items. The effect is commonly created by VR headsets consisting of a head-mounted display with a small screen in front of the eyes, but can also be created through specially designed rooms with multiple large screens. Virtual reality typically incorporates auditory and video feedback, but may also allow other types of sensory and force feedback through haptic technology.

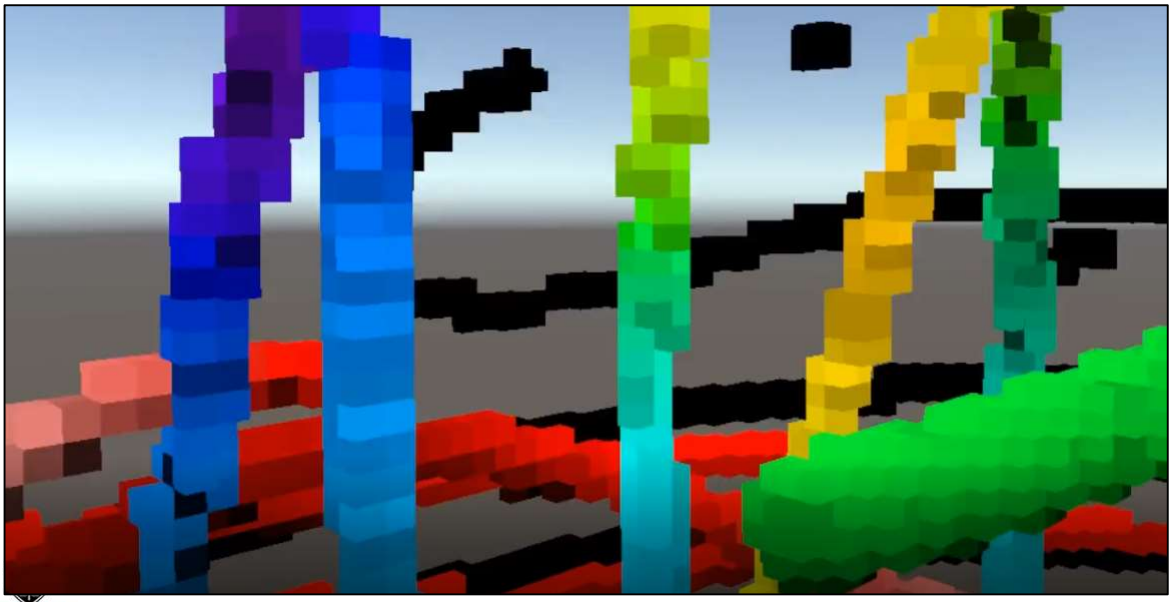
https://en.wikipedia.org/wiki/Virtual_reality

VR Headsets

Uses shaders to get the correct non-linear fisheye lens distortion

Uses an accelerometer and a gyroscope to determine the head position and the head orientation

http://theriftarcade.com



PolyVox, a VR paint program
OSU CS Capstone Project: Richard Cunard, Braxton Cuneo, Chris Bakkom

Inexpensive VR Viewers for your Cell phone

5

Uses OpenGL-ES shaders to get the correct non-linear fisheye lens distortion

Uses your phone's gyroscope to know the head orientation

Aquarium game, SideKick

Uses a moving magnet and the phone's digital compass to perform a "left-click"

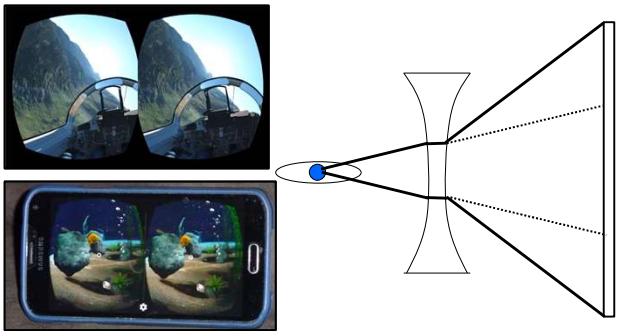
Oregon State University Computer Graphics

mjb – September 9, 2024

5

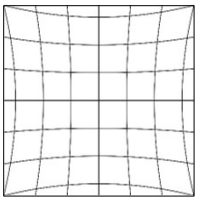
Correcting Lens Distortion with Shader Un-distortion

6



VR/AR goggles use **fisheye lenses** so that your eye can be close to the view screen but still see the whole scene.

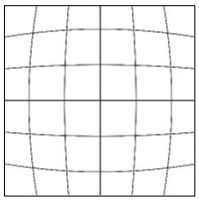
If you displayed a normal CG scene through such a lens, a grid would come out looking like this:



This is known as a **pincushion**.

https://en.wikipedia.org/wiki/Distortion_(optics)

The trick then, is to *distort* the CG image like this so that the lens's pincushion effect will make it look normal:



This is known as a **barrel**.

Oregon State University Computer C

mjb – September 9, 2024

6

Correcting Lens Distortion with Shader Un-distortion

Pass #2 vertex shader

```
#version 330 compatibility
out vec2 vST;

void
main( )
{
	vST = gl_MultiTexCoords0.st;
	gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

Pass #2 fragment shader

```
#version 330 compatibility
uniform sampler2D renderedImage;
in vec2 vST;
const float ALPHA = 0.20;

void
main( )
{
	vec2 xy = 2.*vST – 1.;		// convert [0.,1.] to [-1.,1.]
	xy /= (1. – ALPHA*length(xy) );
	vec2 st = ( xy + 1. ) / 2.;		// convert [-1.,1.] to [0.,1.]
	gl_FragColor = vec4(  texture( renderedImage, st ).rgb, 1.  );
}
```

Use a two-pass rendering algorithm:

Pass #1: Render the image into a texture (not shown here)

Pass #2: Draw a quadrilateral. Normally you would just lookup the rendered texture and map it to the quadrilateral. In this case, we apply a distortion as part of the texture lookup. These are the Pass #2 shaders for drawing the quadrilateral.

There are many different distortion functions you could use. This is one of the simplest.

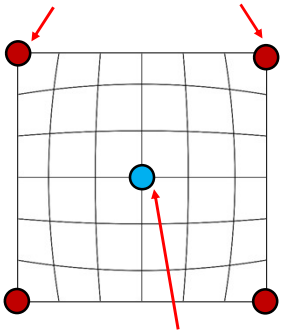
$$Distortion\ Scale\ Factor(xy) = \frac{1}{1 - \alpha * \|(xy)\|}$$

Oregon State University
Computer Graphics

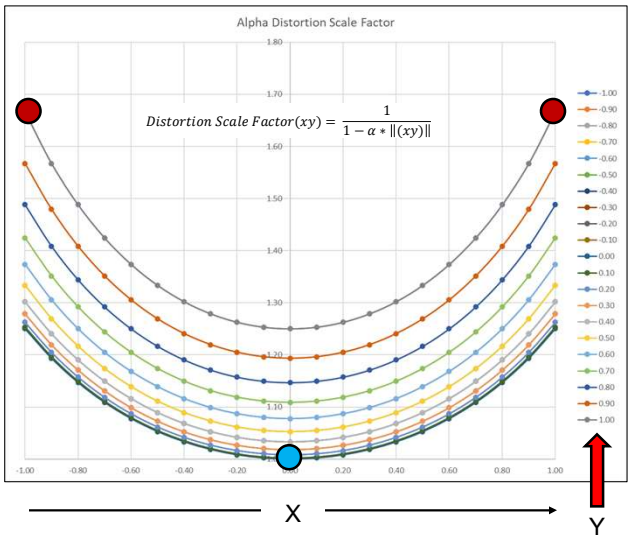
Correcting Lens Distortion with Shader Un-distortion

$$\alpha = 0.20$$

X=±1, Y=±1: most distortion



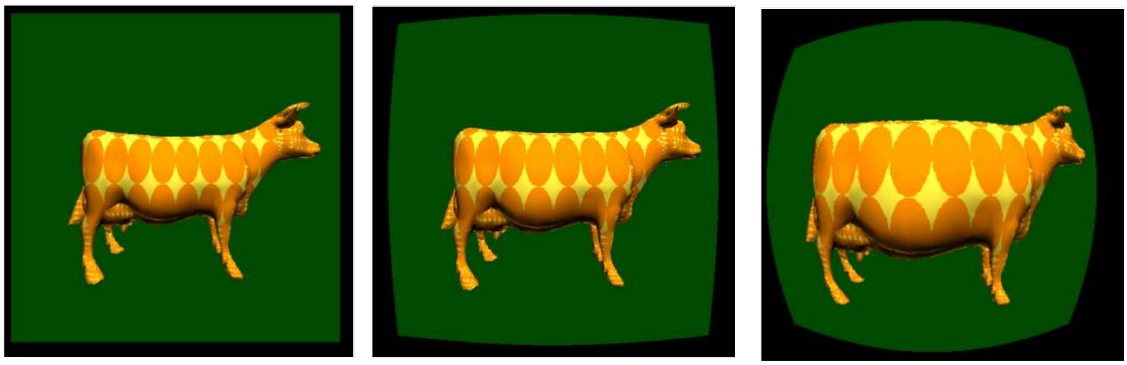
X=0, Y=0: least distortion

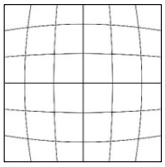


Oregon State University
Computer Graphics

Correcting Lens Distortion with Shader Un-distortion

$\alpha = 0.00$ $\alpha = 0.20$ $\alpha = 1.00$



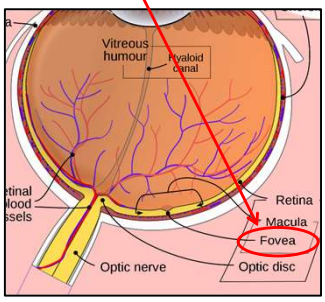


Oregon State University
Computer Graphics

Another Cool VR/AR Rendering Trick – Foveated Rendering

The *fovea* on your retina corresponds to the projection of where you are looking. Of your entire ~220° field-of-view, only the ~2° of the scene projected onto your fovea is really perceived as crisp.

Cool CG trick: track the eye and only render at full resolution the part of the scene where the eye is looking. This saves rendering time and increases graphics performance.

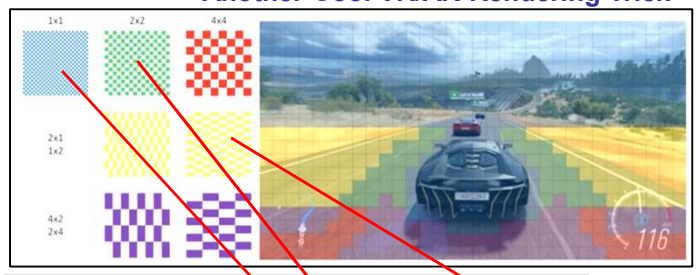


https://en.wikipedia.org/wiki/Fovea_centralis

tobii.com

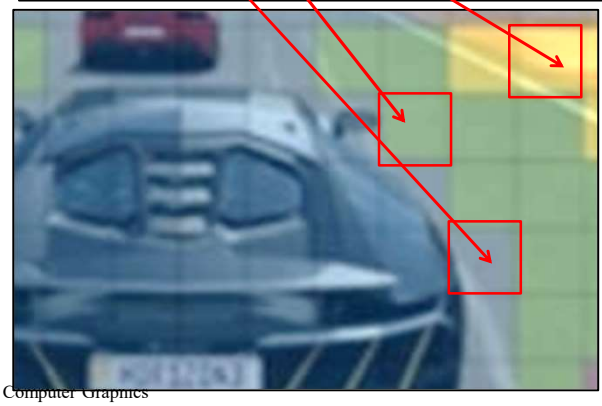
Oregon State University
Computer Graphics

Another Cool VR/AR Rendering Trick – Variable Rate Rendering



By default, there is one fragment shader call per pixel being viewed. The Vulkan API takes foveated rendering and adds more flexibility to it.

Vulkan has a mechanism to enable your program to ask for a fragment shader call to cover more than one pixel. That is, multiple pixels can share the color result from one fragment shader call. This saves rendering time in less important parts of the scene.



Variable Rate Shading can be controlled by specifying parameters for:

1. The entire scene, or,
2. An entire object that is currently being drawn, or,
3. Arbitrary parts of the scene (such as where the eye is looking)

To specify VRS for arbitrary parts of the scene, you create an additional frame buffer and use it to store "flags" in each of its pixels to tell the shading mechanism how many pixels will share the results of a call to that fragment shader.

Inexpensive VR Viewers for your Cell phone

View-Master Virtual Reality

This is the Mattel **View-Master Deluxe VR Viewer.** It sells for under $25. I trust View-Master to get the mechanical design and the optics right. They've been doing this successfully for decades.

BNEXT Virtual Reality

This is the **BNEXT VR Headset.** It sells for under $30. This one is nice because it attaches to your head so that your hands are free. It also has a couple of nice eye-viewing adjustment knobs.

I found both of these on Amazon

Surround-VR: The CAVE

A **Cave Automatic Virtual Environment** (better known by the recursive acronym **CAVE**) is an immersive virtual reality environment where projectors are directed to between three and six of the walls of a room-sized cube.

https://en.wikipedia.org/wiki/Cave_automatic_virtual_environment

https://www.mechdyne.com

Oregon State University
Computer Graphics

Augmented Reality Definition

Augmented reality (**AR**) is an interactive experience of a real-world environment where the objects that reside in the real world are enhanced by computer-generated perceptual information, sometimes across multiple sensory modalities, including visual, auditory, haptic, somatosensory and olfactory. AR can be defined as a system that fulfills three basic features: a combination of real and virtual worlds, real-time interaction, and accurate 3D registration of virtual and real objects. The overlaid sensory information can be constructive (i.e. additive to the natural environment), or destructive (i.e. masking of the natural environment).This experience is seamlessly interwoven with the physical world such that it is perceived as an immersive aspect of the real environment. In this way, augmented reality alters one's ongoing perception of a real-world environment, whereas virtual reality completely replaces the user's real-world environment with a simulated one.

https://en.wikipedia.org/wiki/Augmented_reality

Oregon State University
Computer Graphics

Augmented Reality -- Pokémon Go

15

Pokemon.com

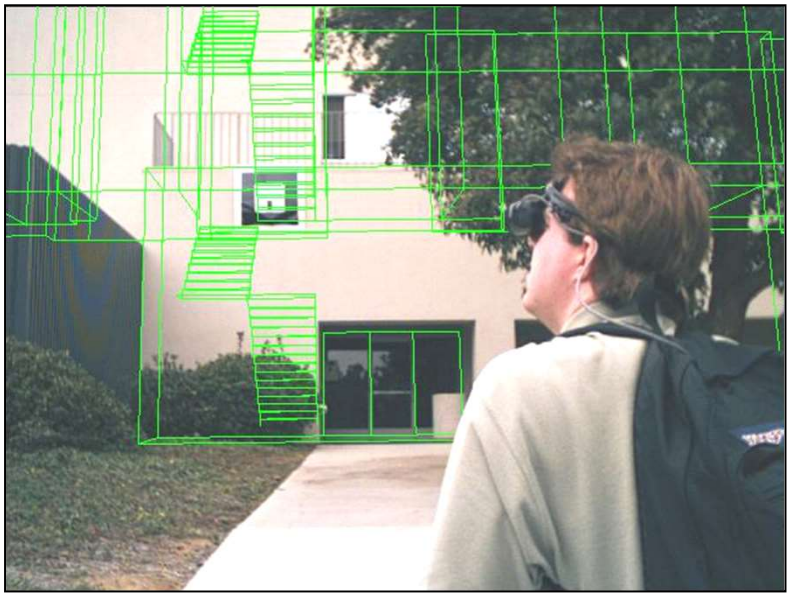
Oregon State University
Computer Graphics

mjb – September 9, 2024

15

Architectural Augmented Reality

16



Matt Clothier

Oregon State University
Computer Graphics

mjb – September 9, 2024

16

High-end Augmented Reality -- Microsoft Hololens 2

https://www.wired.com/story/microsoft-hololens-2-headset/

Microsoft Hololens 2

The lasers in the HoloLens 2 shine into a set of mirrors that oscillate as quickly as 54,000 times per second so the reflected light can paint a display. Those two pieces together form the basis of a microelectromechanical system (MEMS) display. That's all tricky to make, but the really tricky part for a MEMS display is getting the image that it paints into your eyeball.

The Hololens uses ***waveguides***, pieces of glass in front of your eye that are carefully etched so they can reflect the 3D displays. When you put the whole system together — the lasers, the mirrors, and the waveguide — you get a bright display with a wide field of view that doesn't have to be precisely aimed into your eyes to work.

The internal processor is an ARM-based Qualcomm Snapdragon 850, which is designed to be very battery-efficient.

Microsoft Hololens 2 Components

19

Microsoft

Oregon State University Computer Graphics

Microsoft Hololens 2 used to Guide Mechanical Assembly Operations

20

Microsoft

Oregon State University Computer Graphics

Extended Reality

Extended reality (*XR*) is a term referring to all real-and-virtual combined environments and human-machine interactions generated by computer technology and wearables, where the 'X' represents a variable for any current or future spatial computing technologies. It includes representative forms such as augmented reality (AR), mixed reality (MR), and virtual reality (VR) and the areas interpolated among them. The levels of virtuality range from partially sensory inputs to immersive virtuality, also called VR.

XR is a superset which includes the entire spectrum from "the complete real" to "the complete virtual" in the concept of reality–virtuality continuum ... Still, its connotation lies in the extension of human experiences especially relating to the senses of existence (represented by VR) and the acquisition of cognition (represented by AR). With the continuous development in human–computer interactions, this connotation is still evolving.

https://en.wikipedia.org/wiki/Extended_reality

Oregon State University
Computer Graphics

Definitions of Mixed Reality and Augmented Virtuality

Mixed Reality (**MR**) is the merging of real and virtual worlds to produce new environments and visualizations, where physical and digital objects co-exist and interact in real time. Mixed reality does not exclusively take place in either the physical or virtual world, but is a hybrid of reality and virtual reality, encompassing both augmented reality and augmented virtuality via immersive technology.

https://en.wikipedia.org/wiki/Mixed_reality

Augmented Virtuality (**AV**) is a subcategory of mixed reality that refers to the merging of real-world objects into virtual worlds. As an intermediate case in the virtuality continuum, it refers to predominantly virtual spaces, where physical elements (such as physical objects or people) are dynamically integrated into and can interact with the virtual world in real time. This integration is achieved with the use of various techniques, such as streaming video from physical spaces, like through a webcam, or using the 3D digitalization of physical objects.

The use of real-world sensor information, such as gyroscopes, to control a virtual environment is an additional form of augmented virtuality, in which external inputs provide context for the virtual view.

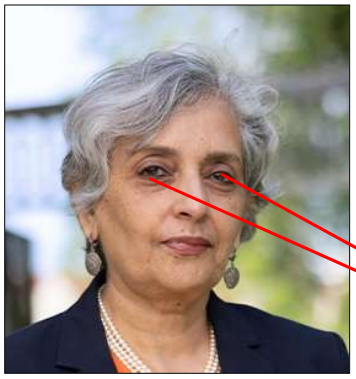
https://en.wikipedia.org/wiki/Mixed_reality

Oregon State University
Computer Graphics

VR/AR Usually Involves Binocular Vision, but Doesn't Have To

In everyday living, part of our perception of depth comes from the slight difference in how our two eyes see the world around us. This is known as *binocular vision*.

We care about this, and are discussing it, because stereo computer graphics can be a great help in de-cluttering a complex 3D scene. It can also enhance the feeling of being immersed in a movie.



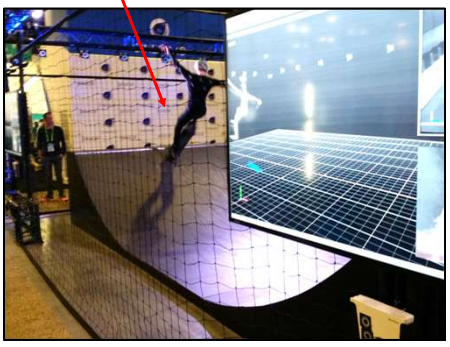
OSU's 16th President Dr. Jayathi Murthy

Oregon State University Computer Graphics

Tracking – Knowing where your Head is and How it is Oriented

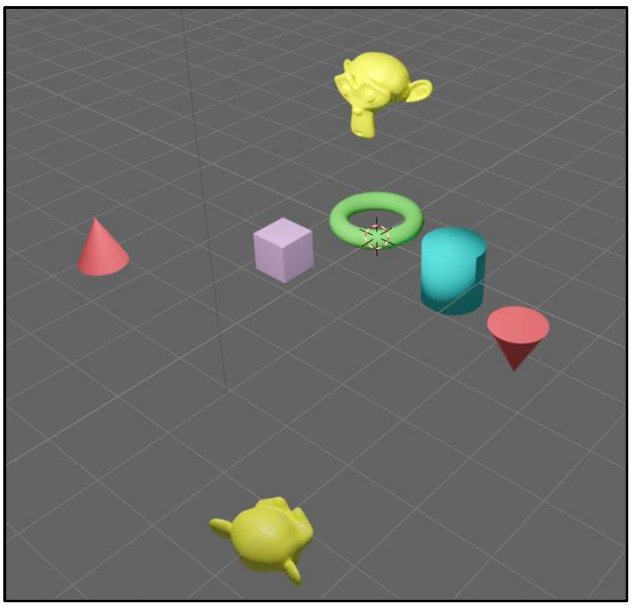
3D Tracking Possibilities:

- Mechanical linkages
- Accelerometers and gyroscopes
- Motion Capture ("MoCap")
- Electromagnetic trackers



Oregon State University Computer Graphics

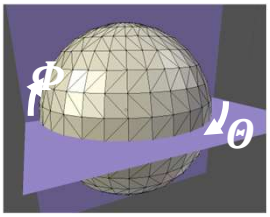
VR with Head Tracking is Good for Walking Through a 3D Scene, Even Without Stereographics

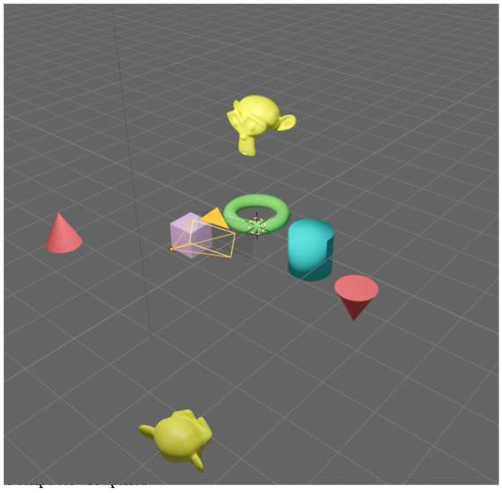


Oregon State University
Computer Graphics

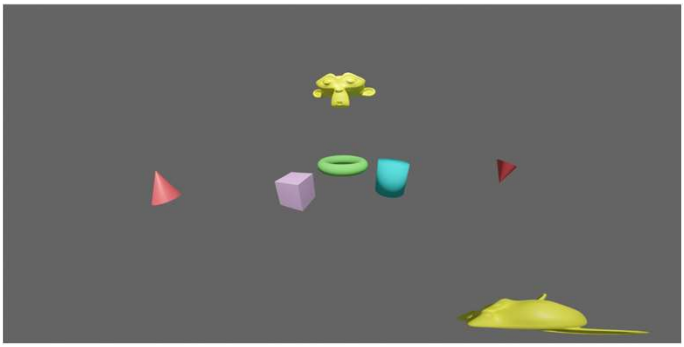
You Can Also Use VR to View a 360° 2D Spherical Image of the Scene

1. Put the eye/camera in the middle of the scene.
2. Render the scene, a vertical strip at a time, onto the inside of a sphere.
3. Save the image, where Θ→s and Φ→t



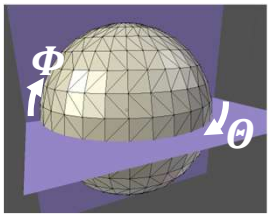


Φ

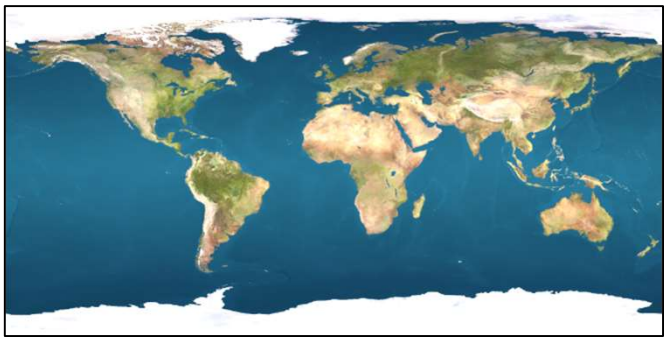


Θ

It's Roughly the Same Idea as Mapping a Texture Image onto a Sphere, but it's the Reverse Process of Mapping a Sphere onto an Image



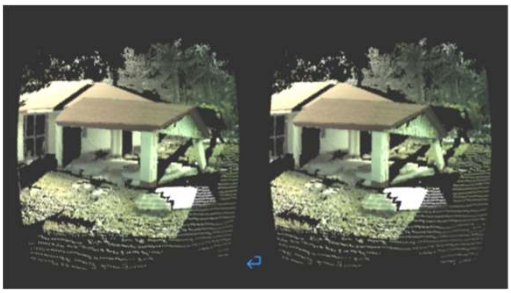
Θ→s
Φ→t

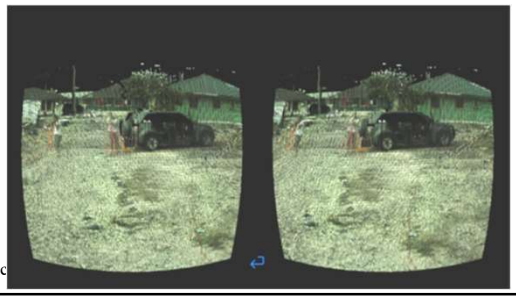


How to Create a 2D Spherical Image of a 3D Scene

```
void  DrawAndWriteSegments( )
{
	unsigned char array[3*PIXELS_PER_SEG*HEIGHT];
	glViewport( 0, 0, PIXELS_PER_SEG, HEIGHT );
	glMatrixMode( GL_PROJECTION );
	glLoadIdentity( );
	gluPerspective( PHIDEG, ASPECT_Y_OVER_X, ZNEAR, ZFAR );
	int col = 0;			// column in the full array
	for( int lookDeg = -90.; lookDeg < 270; lookDeg += PHIDEG )
	{
		glMatrixMode( GL_MODELVIEW );
		glLoadIdentity( );
		float lx = Sind( (float)lookDeg )  +  EX;
		float ly = 0. + EY;
		float lz = Cosd( (float)lookDeg )  +  EZ;
		gluLookAt( EX, EY, EZ,  lx, ly, lz,  0., 1., 0. );
		glCallList( LidarList );
		glFlush( );
		glutSwapBuffers( );
		glFinish( );

		glPixelStorei( GL_PACK_ALIGNMENT, 1 );
		glReadPixels( 0, 0, PIXELS_PER_SEG, HEIGHT, GL_RGB, GL_UNSIGNED_BYTE, array );

		for( int y = 0; y < HEIGHT; y++ )
		{
			memcpy( &FullArray[3*col*HEIGHT+y], &array[3*y*PIXELS_PER_SEG], 3*PIXELS_PER_SEG );
		}
		col += PIXELS_PER_SEG;
	}

	WriteArray( (char *)"Middle.bmp", FullArray );
}
```

You Can Also Do the Spherical Image Projection *with* Stereographics





Oregon State University Computer Graphic

Louis Panton

How to Create Two 2D Spherical Stereographics Images of a 3D Scene

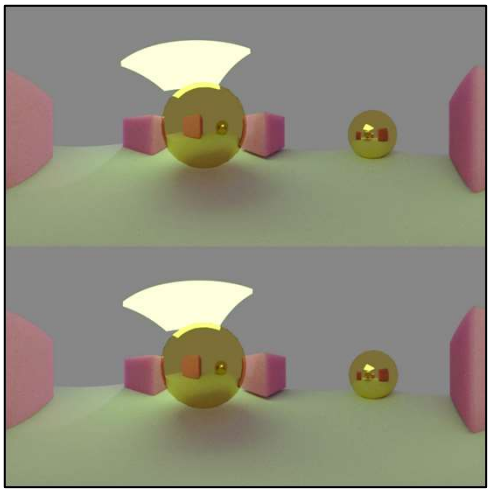
```
void  DrawAndWriteSegments( )
{
	unsigned char array[3*PIXELS_PER_SEG*HEIGHT];
	for( int eye = 0; eye <= 1; eye++ )
	{
		glViewport( 0, 0, PIXELS_PER_SEG, HEIGHT );
		glMatrixMode( GL_PROJECTION );
		glLoadIdentity( );
		StereoPersp( PHIDEG, ASPECT_Y_OVER_X, ZNEAR, ZFAR, Z0P, eye == 0 ? -EYESEP : EYESEP );
		unsigned char *FullArray = ( eye == 0 ? Left : Right );
		int col = 0;			// column in the full array
		for( int lookDeg = -90.; lookDeg < 270; lookDeg += PHIDEG )
		{
			glMatrixMode( GL_MODELVIEW );
			glLoadIdentity( );
			float lx = Sind( (float)lookDeg )  +  EX;
			float ly = 0. + EY;
			float lz = Cosd( (float)lookDeg )  +  EZ;
			gluLookAt( EX, EY, EZ,  lx, ly, lz,  0., 1., 0. );
			glCallList( LidarList );
			glFlush( );
			glutSwapBuffers( );
			glFinish( );

			glPixelStorei( GL_PACK_ALIGNMENT, 1 );
			glReadPixels( 0, 0, PIXELS_PER_SEG, HEIGHT, GL_RGB, GL_UNSIGNED_BYTE, array );

			for( int y = 0; y < HEIGHT; y++ )
			{
				memcpy( &FullArray[3*col*HEIGHT+y], &array[3*y*PIXELS_PER_SEG], 3*PIXELS_PER_SEG );
			}
			col += PIXELS_PER_SEG;
		}

		WriteArray( eye == 0 ? (char *)"Left.bmp" : (char *)"Right.bmp", FullArray );
	}
}
```

Oregon State University Computer Grap

Examples of Spherical Stereographics Images, Suitable for Displaying in a VR Headset

Left Eye View

Right Eye View



Blender Scene

Left Eye View

Right Eye View

ParaView Fluid Flow Streamtubes

Examples of Spherical Stereographics Images, Suitable for Displaying in a VR Headset

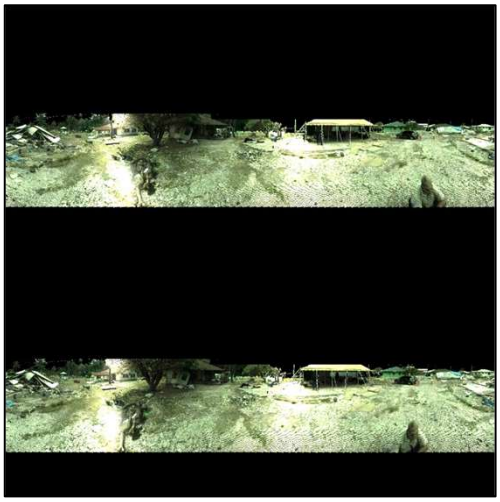
Left Eye View

Right Eye View

My Side Yard
(using a Garmin VIRB 360° camera)

Left Eye View

Right Eye View



Lidar Scene

Data from Dr. Michael Olsen, OSU CCE

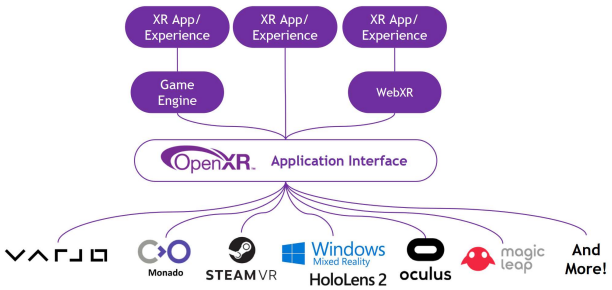
If you are interested in 360° photography, check out the new Insta360 X4

Something to Cut Through the VR/AR "Clutter" -- The Khronos Group's OpenXR API

OpenXR is a royalty-free, open standard that provides high-performance access to Augmented Reality (AR) and Virtual Reality (VR)—collectively known as XR—platforms and devices.

https://www.khronos.org/openxr/

OpenXR Provides a Common API Interface for XR the same way that OpenGL does for 3D Graphics



Oregon State University Computer Graphics

https://www.khronos.org/openxr/

Who's Been Involved with Creating OpenXR?

Oregon State University Computer Graphics

https://www.khronos.org/openxr/