

OpenGL

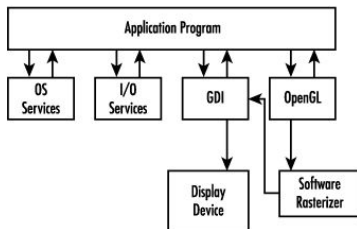
Frederic Koriche

LIRMM, Université Montpellier II

Parcours Intelligence Artificielle
Master Informatique 2ème Année
Septembre 2009

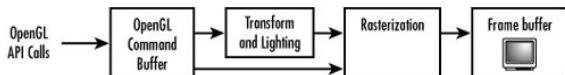


What is OpenGL?



OpenGL is *not* a language. It is an *application programming interface* (API)

How does OpenGL work?



OpenGL works as a *state machine*. OpenGL functions are used to:

- *specify* an initial state,
- *change* from state to state,
- *access* to the current state's values

What is OpenGL environment?

Environment

- Core library: in C language
- Utility library: GLU for higher-order functions (camera control, curves, surfaces, ...)
- Graphic manager: GLUT for managing windows, events, and rendering contexts

Windows Headers

```
#include <windows.h>           // Must have for Windows platform builds
#include "glee.h"              // OpenGL Extension "autoloader"
#include <gl\gl.h>             // Microsoft OpenGL headers (version 1.1 by themselves)
#include <gl\glu.h>            // OpenGL Utilities
#include "glut.h"              // Glut (Free-Glut on Windows)
```

Mac OS X Headers

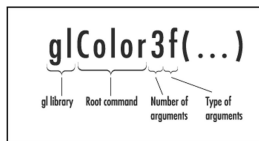
```
#include <Carbon/Carbon.h>     // Brings in most Apple specific stuff
#include "glee.h"              // OpenGL Extension "autoloader"
#include <OpenGL/gl.h>         // Apple OpenGL shaders
#include <OpenGL/glu.h>        // OpenGL Utilities
#include <Glut/glut.h>         // Apples Implementation of GLUT
```

Data Types

OpenGL Data Type	Internal Representation	Defined as C Type	C Literal Suffix
GLbyte	8-bit integer	signed char	b
GLshort	16-bit integer	short	s
GLint, GLsizei	32-bit integer	long	l
GLfloat, GLclampf	32-bit floating point	float	f
GLdouble, GLclampd	64-bit floating point	double	d
GLubyte, GLboolean	8-bit unsigned integer	unsigned char	ub
GLushort	16-bit unsigned integer	unsigned short	us
GLuint, GLenum, GLbitfield	32-bit unsigned integer	unsigned long	ui
GLchar	8-bit character	char	None
GLsizeiPtr, GLintptr	native pointer	ptrdiff_t	None

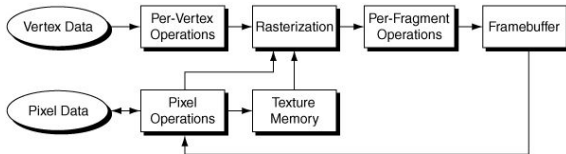
Functions

Function-Naming Conventions



- Library prefix: `gl`
- Root command: `Color`
- Optional Argument Count: `3`
- Optional Argument Type: `f`

Function Types



Colorizing a Window

```
#include "../shared/gltools.h"    // OpenGL toolkit

// Called to draw scene
void RenderScene(void)
{
    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT);

    // Flush drawing commands
    glFlush();
}

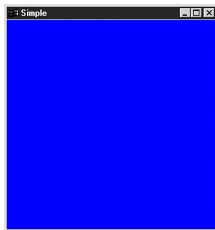
// Set up the rendering state
void SetupRC(void)
{
    // R G B A
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

// Main program entry point
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutCreateWindow("Simple");
    glutDisplayFunc(RenderScene);

    SetupRC();

    glutMainLoop();

    return 0;
}
```



Colorizing a Window

```
#include "../shared/gltools.h"    // OpenGL toolkit

// Called to draw scene
void RenderScene(void)
{
    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT);

    // Flush drawing commands
    glFlush();
}

// Set up the rendering state
void SetupRC(void)
{
    // R G B A
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

// Main program entry point
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutCreateWindow("Simple");
    glutDisplayFunc(RenderScene);

    SetupRC();

    glutMainLoop();

    return 0;
}
```

Main Function

`int main(int argc, char* argv[])` Console mode C program

Display Mode

`glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA)`

- Single-buffered window (no animation)
- RGBA color

Creating Window

`glutCreateWindow("Simple")`

Display Callback

`glutDisplayFunc(RenderScene)`

Rendering Context

`SetupRC()`

Main Loop

`glutMainLoop()`

Colorizing a Window

```
#include "../shared/gltools.h"    // OpenGL toolkit

// Called to draw scene
void RenderScene(void)
{
    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT);

    // Flush drawing commands
    glFlush();
}

// Set up the rendering state
void SetupRC(void)
{
    // R G B A
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

// Main program entry point
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutCreateWindow("Simple");
    glutDisplayFunc(RenderScene);

    SetupRC();

    glutMainLoop();

    return 0;
}
```

Main Function

`int main(int argc, char* argv[])` Console mode C program

Display Mode

`glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA)`

- Single-buffered window (no animation)
- RGBA color

Creating Window

`glutCreateWindow("Simple")`

Display Callback

`glutDisplayFunc(RenderScene)`

Rendering Context

`SetupRC()`

Main Loop

`glutMainLoop()`

Colorizing a Window

```
#include "../shared/gltools.h"    // OpenGL toolkit

// Called to draw scene
void RenderScene(void)
{
    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT);

    // Flush drawing commands
    glFlush();
}

// Set up the rendering state
void SetupRC(void)
{
    // R G B A
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

// Main program entry point
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutCreateWindow("Simple");
    glutDisplayFunc(RenderScene);

    SetupRC();

    glutMainLoop();

    return 0;
}
```

Main Function

`int main(int argc, char* argv[])` Console mode C program

Display Mode

`glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA)`

- Single-buffered window (no animation)
- RGBA color

Creating Window

`glutCreateWindow("Simple")`

Display Callback

`glutDisplayFunc(RenderScene)`

Rendering Context

`SetupRC()`

Main Loop

`glutMainLoop()`

Colorizing a Window

```
#include "../shared/gltools.h"    // OpenGL toolkit

// Called to draw scene
void RenderScene(void)
{
    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT);

    // Flush drawing commands
    glFlush();
}

// Set up the rendering state
void SetupRC(void)
{
    // R G B A
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

// Main program entry point
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutCreateWindow("Simple");
    glutDisplayFunc(RenderScene);

    SetupRC();

    glutMainLoop();

    return 0;
}
```

Main Function

`int main(int argc, char* argv[])` Console mode C program

Display Mode

`glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA)`

- Single-buffered window (no animation)
- RGBA color

Creating Window

`glutCreateWindow("Simple")`

Display Callback

`glutDisplayFunc(RenderScene)`

Rendering Context

`SetupRC()`

Main Loop

`glutMainLoop()`

Colorizing a Window

```
#include "../shared/gltools.h"    // OpenGL toolkit

// Called to draw scene
void RenderScene(void)
{
    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT);

    // Flush drawing commands
    glFlush();
}

// Set up the rendering state
void SetupRC(void)
{
    // R G B A
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

// Main program entry point
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutCreateWindow("Simple");
    glutDisplayFunc(RenderScene);

    SetupRC();

    glutMainLoop();

    return 0;
}
```

Main Function

`int main(int argc, char* argv[])` Console mode C program

Display Mode

`glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA)`

- Single-buffered window (no animation)
- RGBA color

Creating Window

`glutCreateWindow("Simple")`

Display Callback

`glutDisplayFunc(RenderScene)`

Rendering Context

`SetupRC()`

Main Loop

`glutMainLoop()`

Colorizing a Window

```
#include "../shared/gltools.h"    // OpenGL toolkit

// Called to draw scene
void RenderScene(void)
{
    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT);

    // Flush drawing commands
    glFlush();
}

// Set up the rendering state
void SetupRC(void)
{
    // R G B A
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

// Main program entry point
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutCreateWindow("Simple");
    glutDisplayFunc(RenderScene);

    SetupRC();

    glutMainLoop();

    return 0;
}
```

Main Function

`int main(int argc, char* argv[])` Console mode C program

Display Mode

`glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA)`

- Single-buffered window (no animation)
- RGBA color

Creating Window

`glutCreateWindow("Simple")`

Display Callback

`glutDisplayFunc(RenderScene)`

Rendering Context

`SetupRC()`

Main Loop

`glutMainLoop()`

Colorizing a Window

```
#include "../shared/gltools.h"    // OpenGL toolkit

// Called to draw scene
void RenderScene(void)
{
    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT);

    // Flush drawing commands
    glFlush();
}

// Set up the rendering state
void SetupRC(void)
{
    // R G B A
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

// Main program entry point
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutCreateWindow("Simple");
    glutDisplayFunc(RenderScene);

    SetupRC();

    glutMainLoop();

    return 0;
}
```

Setting Background Color

`glClearColor(GLclampf red, green, blue, alpha)` Sets the color used for clearing the window. Values are in $[0, 1]$.

Clearing

`glClear(GL_COLOR_BUFFER_BIT)` Clears the frame buffer with the previously generated color.

Flushing

`glFlush()` Send the sequence of unexecuted GL commands to the hardware.

Colorizing a Window

```
#include "../shared/gltools.h"    // OpenGL toolkit

// Called to draw scene
void RenderScene(void)
{
    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT);

    // Flush drawing commands
    glFlush();
}

// Set up the rendering state
void SetupRC(void)
{
    // R G B A
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

// Main program entry point
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutCreateWindow("Simple");
    glutDisplayFunc(RenderScene);

    SetupRC();

    glutMainLoop();

    return 0;
}
```

Setting Background Color

`glClearColor(GLclampf red, green, blue, alpha)` Sets the color used for clearing the window. Values are in $[0, 1]$.

Clearing

`glClear(GL_COLOR_BUFFER_BIT)` Clears the frame buffer with the previously generated color.

Flushing

`glFlush()` Send the sequence of unexecuted GL commands to the hardware.

Colorizing a Window

```
#include "../shared/gltools.h"    // OpenGL toolkit

// Called to draw scene
void RenderScene(void)
{
    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT);

    // Flush drawing commands
    glFlush();
}

// Set up the rendering state
void SetupRC(void)
{
    // R G B A
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

// Main program entry point
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutCreateWindow("Simple");
    glutDisplayFunc(RenderScene);

    SetupRC();

    glutMainLoop();

    return 0;
}
```

Setting Background Color

`glClearColor(GLclampf red, green, blue, alpha)` Sets the color used for clearing the window. Values are in $[0, 1]$.

Clearing

`glClear(GL_COLOR_BUFFER_BIT)` Clears the frame buffer with the previously generated color.

Flushing

`glFlush()` Send the sequence of unexecuted GL commands to the hardware.

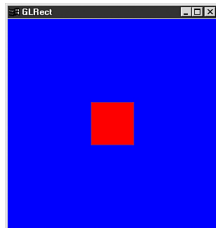
Drawing Shapes

```
void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 0.0f, 0.0f);
    glRectf(-25.0f, 25.0f, 25.0f, -25.0f);
    glFlush();
}

void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

void ChangeSize(GLsizei w, GLsizei h)
{
    GLfloat aspectRatio;
    if(h == 0) h = 1;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    aspectRatio = (GLfloat)w / (GLfloat)h;
    if (w <= h)
        glOrtho (-100.0, 100.0, -100 / aspectRatio,
                100.0 / aspectRatio, 1.0, -1.0);
    else
        glOrtho (-100.0 * aspectRatio, 100.0 * aspectRatio,
                -100.0, 100.0, 1.0, -1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void main(void)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("GLRect");
    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
    SetupRC();
    glutMainLoop();
}
```



Drawing Shapes

```

void RenderScene(void)
{
    glClearColor(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 0.0f, 0.0f);
    glRectf(-25.0f, 25.0f, 25.0f, -25.0f);
    glFlush();
}

void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

void ChangeSize(GLsizei w, GLsizei h)
{
    GLfloat aspectRatio;
    if(h == 0) h = 1;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    aspectRatio = (GLfloat)w / (GLfloat)h;
    if (w <= h)
        glOrtho (-100.0, 100.0, -100 / aspectRatio,
                100.0 / aspectRatio, 1.0, -1.0);
    else
        glOrtho (-100.0 * aspectRatio, 100.0 * aspectRatio,
                -100.0, 100.0, 1.0, -1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void main(void)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("GLRect");
    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
    SetupRC();
    glutMainLoop();
}

```

Drawing a Rectangle

- select color: `glColor3f(R,G,B)`
- draw rectangle: `glRectf(x1, y1, x2, y2)`

(x_1, y_1) is the upper-left corner, and (x_2, y_2) is the lower-right corner.

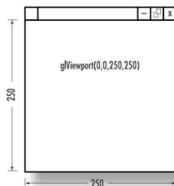
Scaling to the Window

Whenever the window sizes changes, two regions must be redefined to keep proportions.

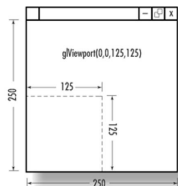
- Clipping: defines the logical volume used by the context to draw objects.
- Viewport: defines physical area used by the context to map pixels.

Defining the Viewport

```
void glViewport(GLint x, y, GLsizei width, height)
```



Window and viewport are same



Viewport 1/2 size of window

Drawing Shapes

```

void RenderScene(void)
{
    glClearColor(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 0.0f, 0.0f);
    glRectf(-25.0f, 25.0f, 25.0f, -25.0f);
    glFlush();
}

void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

void ChangeSize(GLsizei w, GLsizei h)
{
    GLfloat aspectRatio;
    if(h == 0) h = 1;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    aspectRatio = (GLfloat)w / (GLfloat)h;
    if (w <= h)
        glOrtho (-100.0, 100.0, -100 / aspectRatio,
                100.0 / aspectRatio, 1.0, -1.0);
    else
        glOrtho (-100.0 * aspectRatio, 100.0 * aspectRatio,
                -100.0, 100.0, 1.0, -1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void main(void)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("GLRect");
    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
    SetupRC();
    glutMainLoop();
}

```

Drawing a Rectangle

- select color: `glColor3f(R,G,B)`
- draw rectangle: `glRectf(x1, y1, x2, y2)`

(x_1, y_1) is the upper-left corner, and (x_2, y_2) is the lower-right corner.

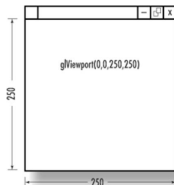
Scaling to the Window

Whenever the window sizes changes, two regions must be redefined to keep proportions.

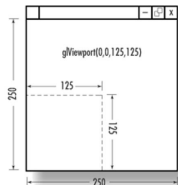
- Clipping: defines the logical volume used by the context to draw objects.
- Viewport: defines physical area used by the context to map pixels.

Defining the Viewport

```
void glViewport(GLint x, y, GLsizei width, height)
```



Window and viewport are same



Viewport 1/2 size of window

Drawing Shapes

```

void RenderScene(void)
{
    glClearColor(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 0.0f, 0.0f);
    glRectf(-25.0f, 25.0f, 25.0f, -25.0f);
    glFlush();
}

void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

void ChangeSize(GLsizei w, GLsizei h)
{
    GLfloat aspectRatio;
    if(h == 0) h = 1;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    aspectRatio = (GLfloat)w / (GLfloat)h;
    if (w <= h)
        glOrtho (-100.0, 100.0, -100 / aspectRatio,
                100.0 / aspectRatio, 1.0, -1.0);
    else
        glOrtho (-100.0 * aspectRatio, 100.0 * aspectRatio,
                -100.0, 100.0, 1.0, -1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void main(void)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("GLRect");
    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
    SetupRC();
    glutMainLoop();
}

```

Drawing a Rectangle

- select color: `glColor3f(R,G,B)`
- draw rectangle: `glRectf(x1, y1, x2, y2)`

(x_1, y_1) is the upper-left corner, and (x_2, y_2) is the lower-right corner.

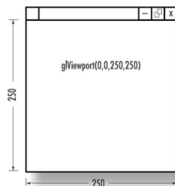
Scaling to the Window

Whenever the window sizes changes, two regions must be redefined to keep proportions.

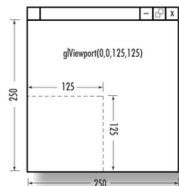
- Clipping: defines the logical volume used by the context to draw objects.
- Viewport: defines physical area used by the context to map pixels.

Defining the Viewport

`void glViewport(GLint x, y, GLsizei width, height)`



Window and viewport are same



Viewport 1/2 size of window

Drawing Shapes

```

void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 0.0f, 0.0f);
    glRectf(-25.0f, 25.0f, 25.0f, -25.0f);
    glFlush();
}

void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

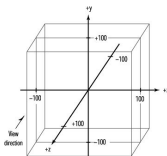
void ChangeSize(GLsizei w, GLsizei h)
{
    GLfloat aspectRatio;
    if(h == 0) h = 1;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    aspectRatio = (GLfloat)w / (GLfloat)h;
    if (w <= h)
        glOrtho (-100.0, 100.0, -100 / aspectRatio,
                100.0 / aspectRatio, 1.0, -1.0);
    else
        glOrtho (-100.0 * aspectRatio, 100.0 * aspectRatio,
                -100.0, 100.0, 1.0, -1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void main(void)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("GLRect");
    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
    SetupRC();
    glutMainLoop();
}

```

Defining an Orthographic Clipped Volume

```
void glOrtho(GLdouble left, right, bottom,
            top, near, far)
```



Keeping a Square Square

`aspectRatio`: the width of the logical window divided by its height.

- if $w \leq h$, the horizontal is 200
- if $w > h$, the horizontal is scaled by the aspect ratio
- if $w \leq h$, the vertical is 200
- if $w > h$, the vertical is scaled by the aspect ratio

Resetting Coordinates

```
void glLoadIdentity()
```

Drawing Shapes

```

void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 0.0f, 0.0f);
    glRectf(-25.0f, 25.0f, 25.0f, -25.0f);
    glFlush();
}

void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

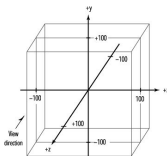
void ChangeSize(GLsizei w, GLsizei h)
{
    GLfloat aspectRatio;
    if(h == 0) h = 1;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    aspectRatio = (GLfloat)w / (GLfloat)h;
    if (w <= h)
        glOrtho (-100.0, 100.0, -100 / aspectRatio,
                100.0 / aspectRatio, 1.0, -1.0);
    else
        glOrtho (-100.0 * aspectRatio, 100.0 * aspectRatio,
                -100.0, 100.0, 1.0, -1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void main(void)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("GLRect");
    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
    SetupRC();
    glutMainLoop();
}

```

Defining an Orthographic Clipped Volume

```
void glOrtho(GLdouble left, right, bottom,
top, near, far)
```



Keeping a Square Square

aspectRatio: the width of the logical window divided by its height.

- if $w \leq h$, the horizontal is 200
- if $w > h$, the horizontal is scaled by the aspect ratio
- if $w \leq h$, the vertical is 200
- if $w > h$, the vertical is scaled by the aspect ratio

Resetting Coordinates

```
void glLoadIdentity()
```

Drawing Shapes

```

void RenderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 0.0f, 0.0f);
    glRectf(-25.0f, 25.0f, 25.0f, -25.0f);
    glFlush();
}

void SetupRC(void)
{
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
}

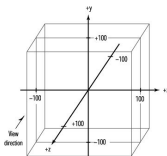
void ChangeSize(GLsizei w, GLsizei h)
{
    GLfloat aspectRatio;
    if(h == 0) h = 1;
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    aspectRatio = (GLfloat)w / (GLfloat)h;
    if (w <= h)
        glOrtho (-100.0, 100.0, -100 / aspectRatio,
                100.0 / aspectRatio, 1.0, -1.0);
    else
        glOrtho (-100.0 * aspectRatio, 100.0 * aspectRatio,
                -100.0, 100.0, 1.0, -1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void main(void)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutCreateWindow("GLRect");
    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
    SetupRC();
    glutMainLoop();
}

```

Defining an Orthographic Clipped Volume

```
void glOrtho(GLdouble left, right, bottom,
top, near, far)
```



Keeping a Square Square

aspectRatio: the width of the logical window divided by its height.

- if $w \leq h$, the horizontal is 200
- if $w > h$, the horizontal is scaled by the aspect ratio
- if $w \leq h$, the vertical is 200
- if $w > h$, the vertical is scaled by the aspect ratio

Resetting Coordinates

```
void glLoadIdentity()
```

Animating Shapes

```

// Initial square position and size
GLfloat x1 = 0.0f;
GLfloat y1 = 0.0f;
GLfloat rsize = 25;

// Step size in x and y directions
GLfloat xstep = 1.0f;
GLfloat ystep = 1.0f;

// Keep track of windows changing width and height
GLfloat windowHeight;
GLfloat windowHeight;

// Called to draw scene
void RenderScene(void)
{
    // Clear the window with current clearing color
    glClearColor(GL_COLOR_BUFFER_BIT);

    // Set current drawing color to red
    glColor3f(1.0f, 0.0f, 0.0f);

    // Draw a filled rectangle with current color
    glRectf(x1, y1, x1 + rsize, y1 - rsize);

    // Flush drawing commands and swap
    glutSwapBuffers();
}

// Called by GLUT library when idle
void TimerFunction(int value)
{
    // Reverse direction when you reach left or right edge
    if(x1 > windowHeight-rsize || x1 < -windowWidth)
        xstep = -xstep;

    // Reverse direction when you reach top or bottom edge
    if(y1 > windowHeight || y1 < -windowHeight + rsize)
        ystep = -ystep;

    // Actually move the square
    x1 += xstep;
    y1 += ystep;

    // Check bounds.
    if(x1 > (windowWidth-rsize + xstep))
        x1 = windowHeight-rsize-1;
    else if(x1 < -(windowWidth + xstep))
        x1 = -windowWidth -1;

    if(y1 > (windowHeight + ystep))
        y1 = windowHeight-1;
    else if(y1 < -(windowHeight - rsize + ystep))
        y1 = -windowHeight + rsize -1;

    // Redraw the scene with new coordinates
    glutPostRedisplay();
    glutTimerFunc(33,TimerFunction,1);
}

// Main program entry point
void main(void)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(800,600);
    glutCreateWindow("Bounce");
    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
    glutTimerFunc(33, TimerFunction, 1);

    SetupRC();
    glutMainLoop();
}

```

Animating Shapes

```

// Called by GLUT library when idle
void TimerFunction(int value)
{
    // Reverse direction when you reach left or right edge
    if(x1 > windowWidth-rsize || x1 < -windowWidth)
        xstep = -xstep;

    // Reverse direction when you reach top or bottom edge
    if(y1 > windowHeight || y1 < -windowHeight + rsize)
        ystep = -ystep;

    // Actually move the square
    x1 += xstep;
    y1 += ystep;

    // Check bounds.
    if(x1 > (windowWidth-rsize + xstep))
        x1 = windowWidth-rsize-1;
    else if(x1 < -(windowWidth + xstep))
        x1 = - windowWidth -1;

    if(y1 > (windowHeight + ystep))
        y1 = windowHeight-1;
    else if(y1 < -(windowHeight - rsize + ystep))
        y1 = -windowHeight + rsize -1;

    // Redraw the scene with new coordinates
    glutPostRedisplay();
    glutTimerFunc(33,TimerFunction,1);
}

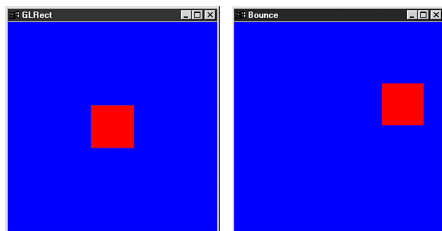
```

Simple Animations

`void glutTimerFunc(uint msec, void (*func)(int value), int value)` Waits msec milliseconds before calling func with parameter value. When the time expires, the function is fired only once. A recursive call is needed for a continuous animation.

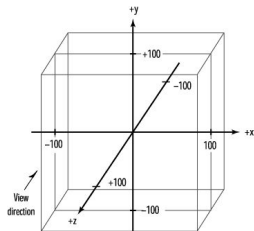
Double Buffering

`glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA)` The image is drawn offscreen and, when it is complete, the image is displayed using `glutSwapBuffers()`.



Drawing Primitives

```
glBegin(GL_POINTS);           // Select points as the primitive
    glVertex3f(0.0f, 0.0f, 0.0f); // Specify a point
    glVertex3f(50.0f, 50.0f, 50.0f); // Specify another point
glEnd();                       // Done drawing points
```



Volume measuring $100 \times 100 \times 100$



GL_POINTS



GL_LINES



GL_LINE_STRIP



GL_LINE_LOOP



GL_TRIANGLES



GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN



GL_QUADS



GL_QUAD_STRIP



GL_POLYGON

Drawing Points

```
// Define a constant for the value of PI
#define GL_PI 3.1415f

// Rendering context.
void SetupRC()
{
    // Black background
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f );

    // Set drawing color to white
    glColor3f(1.0f, 1.0f, 1.0f);
}
```



```
// Called to draw scene
void RenderScene(void)
{
    GLfloat x,y,z,angle; // Storage for coordinates and angles

    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT);

    // Save matrix state and do the rotation
    glPushMatrix();
    glRotatf(xRot, 1.0f, 0.0f, 0.0f);
    glRotatf(yRot, 0.0f, 1.0f, 0.0f);

    // Draw series of points
    glBegin(GL_POINTS);

    z = -50.0f;
    for(angle = 0.0f; angle <= (2.0f*GL_PI)*3.0f; angle += 0.1f)
    {
        x = 50.0f*sin(angle);
        y = 50.0f*cos(angle);

        // Specify the point and move the Z value up a little
        glVertex3f(x, y, z);
        z += 0.5f;
    }

    // Done drawing points
    glEnd();

    // Restore transformations
    glPopMatrix();

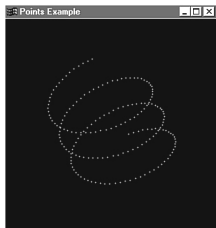
    // Flush drawing commands
    glutSwapBuffers();
}
```

Drawing Points

```
// Define a constant for the value of PI
#define GL_PI 3.1415f

// Rendering context.
void SetupRC()
{
    // Black background
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f );

    // Set drawing color to white
    glColor3f(1.0f, 1.0f, 1.0f);
}
```



```
// Called to draw scene
void RenderScene(void)
{
    GLfloat x,y,z,angle; // Storage for coordinates and angles

    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT);

    // Save matrix state and do the rotation
    glPushMatrix();
    glRotatf(xRot, 1.0f, 0.0f, 0.0f);
    glRotatf(yRot, 0.0f, 1.0f, 0.0f);

    // Draw series of points
    glBegin(GL_POINTS);

    z = -50.0f;
    for(angle = 0.0f; angle <= (2.0f*GL_PI)*3.0f; angle += 0.1f)
    {
        x = 50.0f*sin(angle);
        y = 50.0f*cos(angle);

        // Specify the point and move the Z value up a little
        glVertex3f(x, y, z);
        z += 0.5f;
    }

    // Done drawing points
    glEnd();

    // Restore transformations
    glPopMatrix();

    // Flush drawing commands
    glutSwapBuffers();
}
```

Drawing Points

Setting the Point Size

The size is one pixel by default. To change the size:

```
glPointSize(GLfloat size)
```

To store supported sizes:

```
GLfloat sizes[2];
```

```
glGetFloatv(GL_POINT_SIZE_RANGE,&sizes);
```

To store supported increments:

```
GLfloat step;
```

```
glGetFloatv(GL_POINT_SIZE_GRANULARITY,&step);
```

```
// Called to draw scene
void RenderScene(void)
{
    GLfloat x,y,z,angle; // Storage for coordinates and angles
    GLfloat sizes[2];    // Store supported point size range
    GLfloat step;        // Store supported point size increments
    GLfloat curSize;     // Store current point size
    ...

    // Get supported point size range and step size
    glGetFloatv(GL_POINT_SIZE_RANGE,sizes);
    glGetFloatv(GL_POINT_SIZE_GRANULARITY,&step);

    // Set the initial point size
    curSize = sizes[0];

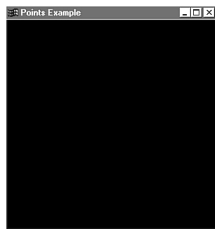
    // Set beginning z coordinate
    z = -50.0f;

    for(angle = 0.0f; angle <= (2.0f*GL_PI)*3.0f; angle += 0.1f)
    {
        // Calculate x and y values on the circle
        x = 50.0f*sin(angle);
        y = 50.0f*cos(angle);

        // Specify the point size before the primitive is specified
        glPointSize(curSize);

        // Draw the point
        glBegin(GL_POINTS);
            glVertex3f(x, y, z);
        glEnd();

        // Bump up the z value and the point size
        z += 0.5f;
        curSize += step;
    }
    ...
}
```



Drawing Points

Setting the Point Size

The size is one pixel by default. To change the size:

```
glPointSize(GLfloat size)
```

To store supported sizes:

```
GLfloat sizes[2];
```

```
glGetFloatv(GL_POINT_SIZE_RANGE,&sizes);
```

To store supported increments:

```
GLfloat step;
```

```
glGetFloatv(GL_POINT_SIZE_GRANULARITY,&step);
```

```
// Called to draw scene
void RenderScene(void)
{
    GLfloat x,y,z,angle; // Storage for coordinates and angles
    GLfloat sizes[2];    // Store supported point size range
    GLfloat step;       // Store supported point size increments
    GLfloat curSize;    // Store current point size
    ...

    // Get supported point size range and step size
    glGetFloatv(GL_POINT_SIZE_RANGE,sizes);
    glGetFloatv(GL_POINT_SIZE_GRANULARITY,&step);

    // Set the initial point size
    curSize = sizes[0];

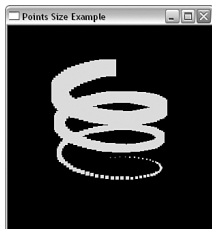
    // Set beginning z coordinate
    z = -50.0f;

    for(angle = 0.0f; angle <= (2.0f*GL_PI)*3.0f; angle += 0.1f)
    {
        // Calculate x and y values on the circle
        x = 50.0f*sin(angle);
        y = 50.0f*cos(angle);

        // Specify the point size before the primitive is specified
        glPointSize(curSize);

        // Draw the point
        glBegin(GL_POINTS);
            glVertex3f(x, y, z);
        glEnd();

        // Bump up the z value and the point size
        z += 0.5f;
        curSize += step;
    }
    ...
}
```



Drawing Lines

Specifying a Line

```
glBegin(GL_LINES);  
    glVertex3f(0.0f, 0.0f, 0.0f);  
    glVertex3f(50.0f, 50.0f, 50.0f);  
glEnd();
```

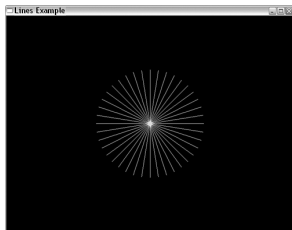
```
// Call only once for all remaining points  
glBegin(GL_LINES);  
  
// All lines lie in the xy plane.  
z = 0.0f;  
for(angle = 0.0f; angle <= GL_PI; angle += (GL_PI/20.0f))  
{  
    // Top half of the circle  
    x = 50.0f*sin(angle);  
    y = 50.0f*cos(angle);  
    glVertex3f(x, y, z);        // First endpoint of line  
  
    // Bottom half of the circle  
    x = 50.0f*sin(angle + GL_PI);  
    y = 50.0f*cos(angle + GL_PI);  
    glVertex3f(x, y, z);        // Second endpoint of line  
}  
  
// Done drawing points  
glEnd();
```



Drawing Lines

Specifying a Line

```
glBegin(GL_LINES);  
    glVertex3f(0.0f, 0.0f, 0.0f);  
    glVertex3f(50.0f, 50.0f, 50.0f);  
glEnd();
```



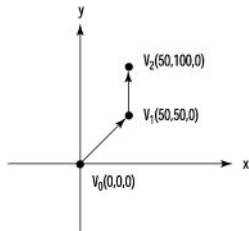
```
// Call only once for all remaining points  
glBegin(GL_LINES);  
  
// All lines lie in the xy plane.  
z = 0.0f;  
for(angle = 0.0f; angle <= GL_PI; angle += (GL_PI/20.0f))  
{  
    // Top half of the circle  
    x = 50.0f*sin(angle);  
    y = 50.0f*cos(angle);  
    glVertex3f(x, y, z);          // First endpoint of line  
  
    // Bottom half of the circle  
    x = 50.0f*sin(angle + GL_PI);  
    y = 50.0f*cos(angle + GL_PI);  
    glVertex3f(x, y, z);        // Second endpoint of line  
}  
  
// Done drawing points  
glEnd();
```

Drawing Lines

Line Strips

Drawing continuous segments.

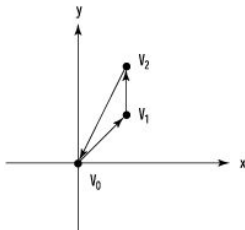
```
glBegin(GL_LINE_STRIP);
  glVertex3f(0.0f, 0.0f, 0.0f);    // V0
  glVertex3f(50.0f, 50.0f, 0.0f); // V1
  glVertex3f(50.0f, 100.0f, 0.0f); // V2
glEnd();
```



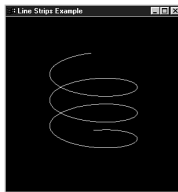
Line Loops

Drawing continuous segments, with one final segment between the last and the first vertices.

```
glBegin(GL_LINE_LOOP);
  glVertex3f(0.0f, 0.0f, 0.0f);    // V0
  glVertex3f(50.0f, 50.0f, 0.0f); // V1
  glVertex3f(50.0f, 100.0f, 0.0f); // V2
glEnd();
```



Curves can be approximated with straight lines, by replacing `GL_POINTS` with `GL_LINE_STRIP`



Drawing Lines

Setting Line Width

To change the width:

```
glLineWidth(GLfloat width)
```

To store supported widths:

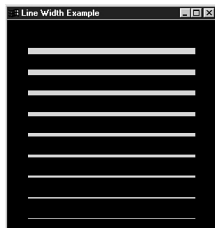
```
GLfloat sizes[2];
```

```
glGetFloatv(GL_LINE_WIDTH_RANGE,&sizes);
```

To store supported increments:

```
GLfloat step;
```

```
glGetFloatv(GL_LINE_WIDTH_GRANULARITY,&step);
```



Line Stippling

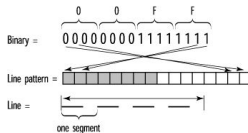
To enable stippling:

```
glEnable(GL_LINE_STIPPLE)
```

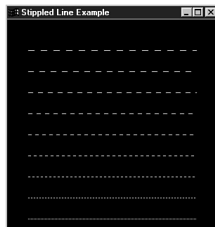
To establish the pattern:

```
glLineStipple(GLint factor, GLushort pattern);
```

Pattern = 0X00FF = 255



Each bit of the pattern becomes factor pixels. An example with pattern 0×5555 and factors [1, 9].



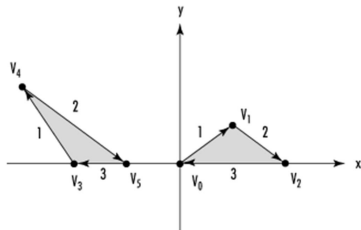
Drawing Triangles

Specifying Triangles

```
glFrontFace(GL_CW);

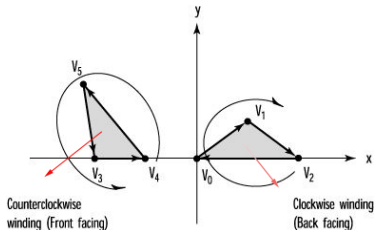
glBegin(GL_TRIANGLES);
  glVertex2f(0.0f, 0.0f);           // V0
  glVertex2f(25.0f, 25.0f);        // V1
  glVertex2f(50.0f, 0.0f);         // V2

  glVertex2f(-50.0f, 0.0f);        // V3
  glVertex2f(-75.0f, 50.0f);       // V4
  glVertex2f(-25.0f, 0.0f);        // V5
glEnd();
```



Polygon Winding

Winding is the order in which the vertices are specified when they are projected on the x - y -plane. By default, OpenGL treats polygons in *counterclockwise winding*.



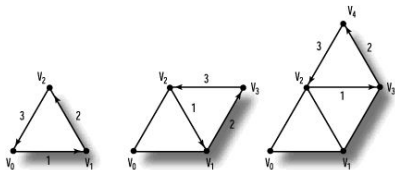
To change the winding:
`glFrontFace(GL_CW)`

Drawing Triangles

Triangle Strips

Stripping preserves the counterclockwise winding. Here, the pattern is (V_0, V_1, V_2) , then (V_2, V_1, V_3) , then (V_2, V_3, V_4) , and so on.

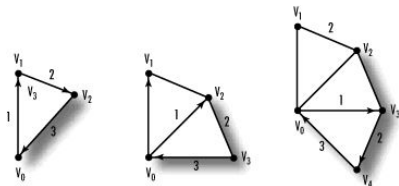
```
glBegin(GL_TRIANGLE_STRIP);
    glVertex3f(0.0f, 0.0f, 0.0f); // V0
    glVertex3f(60.0f, 0.0f, 0.0f); // V1
    glVertex3f(30.0f, 30.0f, 0.0f); // V2
    glVertex3f(90.0f, 30.0f, 0.0f); // V3
    glVertex3f(90.0f, 60.0f, 0.0f); // V4
glEnd();
```



Triangle Fans

Producing a group of connected triangles that fan around a central point.

```
glBegin(GL_TRIANGLE_FAN);
    glVertex2f(0.0f, 0.0f);
    for(angle = 0.0f; angle < (2.0f*GL_PI); angle += (GL_PI/8.0f))
    {
        x = 50.0f*sin(angle);
        y = 50.0f*cos(angle);
        glVertex2f(x, y);
    }
glEnd();
```



```

void SetupRC()
{
    // Black background
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f );
    // Set drawing color to green
    glColor3f(0.0f, 1.0f, 0.0f);
    // Set color shading model to flat
    glShadeModel(GL_FLAT);
    // Clockwise-wound polygons
    glFrontFace(GL_CW);
}

void RenderScene(void)
{
    GLfloat x,y,angle; // Storage for coordinates and angles
    int iPivot = 1;    // Used to flag alternating colors

    // Clear the window and the depth buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Turn culling on
    glEnable(GL_CULL_FACE);

    // Enable depth testing
    glEnable(GL_DEPTH_TEST);

    // Draw the back side as a wireframe only
    glPolygonMode(GL_BACK, GL_LINE);

    // Save matrix state and do the rotation
    glPushMatrix();
    glRotatef(xRot, 1.0f, 0.0f, 0.0f);
    glRotatef(yRot, 0.0f, 1.0f, 0.0f);

    // Begin a triangle fan
    glBegin(GL_TRIANGLE_FAN);

    // move up z-axis to produce a cone instead of a circle
    glVertex3f(0.0f, 0.0f, 75.0f);

    for(angle = 0.0f; angle < (2.0f*GL_PI); angle += (GL_PI/8.0f))
    {
        // Calculate x and y position of the next vertex
        x = 50.0f*sin(angle);
        y = 50.0f*cos(angle);

        // Alternate color between red and green
        if((iPivot %2) == 0)
            glColor3f(0.0f, 1.0f, 0.0f);
        else
            glColor3f(1.0f, 0.0f, 0.0f);

        // Increment pivot to change color next time
        iPivot++;

        // Specify the next vertex for the triangle fan
        glVertex2f(x, y);
    }
    glEnd();

    // Begin a new triangle fan to cover the bottom
    glBegin(GL_TRIANGLE_FAN);

    glVertex2f(0.0f, 0.0f);
    for(angle = 0.0f; angle < (2.0f*GL_PI); angle += (GL_PI/8.0f))
    {
        x = 50.0f*sin(angle);
        y = 50.0f*cos(angle);

        if((iPivot %2) == 0)
            glColor3f(0.0f, 1.0f, 0.0f);
        else
            glColor3f(1.0f, 0.0f, 0.0f);

        iPivot++;

        glVertex2f(x, y);
    }
    glEnd();

    // Restore transformations
    glPopMatrix();
    glutSwapBuffers ();
}

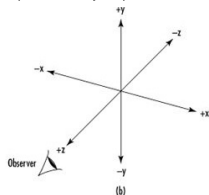
```

Drawing Triangles

Depth Testing

Effective technique for hidden surface removal.

When a pixel is drawn, it is assigned a z value that denotes the distance from the viewer's perspective. Then, pixels at the same position are compared and only the pixel with maximal z value is displayed.



To request a color and a depth buffer:

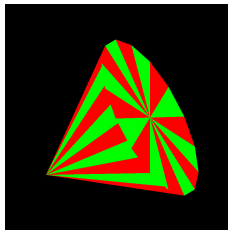
```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
```

To clear color and depth buffers:

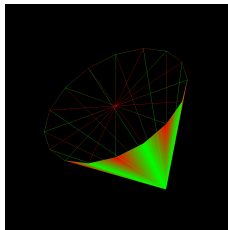
```
glClearColor(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

To enable depth buffer:

```
glEnable(GL_DEPTH_TEST);
```



Rendering of the initial object



Rendering of the object with `GL_SMOOTH` shade model and `GL_LINE` drawing model for the back side

Vertex Arrays

Specifying Arrays

Vertex arrays allow applications to cache blocks of vertices in the GPU.

An array storing small random stars:

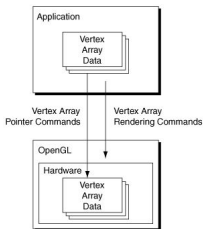
```
#define SMALL_STARS 100
M3DVector2f vSmallStars[SMALL_STARS];
```

Enabling the use of vertex arrays:

```
glEnableClientState(GL_VERTEX_ARRAY);
```

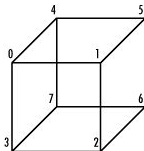
Pointing on the vertex data:

```
glVertexPointer(2, GL_FLOAT, 0, vSmallStars);
```



Indexed Vertex Arrays

A vertex array is often associated with an *index* array which specifies the polygon geometry of the vertex array.



```
// Array containing the six vertices of the cube
static GLfloat corners[] = { -25.0f, 25.0f, 25.0f, // 0 // Front
                             25.0f, 25.0f, 25.0f, // 1
                             25.0f, -25.0f, 25.0f, // 2
                             -25.0f, -25.0f, 25.0f, // 3
                             -25.0f, 25.0f, -25.0f, // 4 // Back
                             25.0f, 25.0f, -25.0f, // 5
                             25.0f, -25.0f, -25.0f, // 6
                             -25.0f, -25.0f, -25.0f }; // 7
```

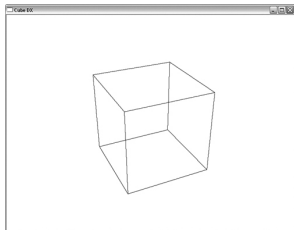
```
// Array of indexes to create the cube
static GLubyte indexes[] = { 0, 1, 2, 3, // Front Face
                              4, 5, 1, 0, // Top Face
                              3, 2, 6, 7, // Bottom Face
                              5, 4, 7, 6, // Back Face
                              1, 5, 6, 2, // Right Face
                              4, 0, 3, 7 }; // Left Face
```

Vertex Arrays

```
// Array containing the six vertices of the cube
static GLfloat corners[] = { -25.0f, 25.0f, 25.0f, // 0
                             25.0f, 25.0f, 25.0f, // 1
                             25.0f, -25.0f, 25.0f, // 2
                             -25.0f, -25.0f, 25.0f, // 3
                             -25.0f, 25.0f, -25.0f, // 4
                             25.0f, 25.0f, -25.0f, // 5
                             25.0f, -25.0f, -25.0f, // 6
                             -25.0f, -25.0f, -25.0f }; // 7
```

```
// Array of indexes to create the cube
static GLubyte indexes[] = { 0, 1, 2, 3, // Front Face
                              4, 5, 1, 0, // Top Face
                              3, 2, 6, 7, // Bottom Face
                              5, 4, 7, 6, // Back Face
                              1, 5, 6, 2, // Right Face
                              4, 0, 3, 7 }; // Left Face
```

```
// Rotation amounts
static GLfloat xRot = 0.0f;
static GLfloat yRot = 0.0f;
```



```
// Called to draw scene
void RenderScene(void)
{
    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Make the cube wireframe
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

    // Save the matrix state
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glTranslatef(0.0f, 0.0f, -200.0f);

    // Rotate about x and y axes
    glRotatef(xRot, 1.0f, 0.0f, 0.0f);
    glRotatef(yRot, 0.0f, 0.0f, 1.0f);

    // Enable and specify the vertex array
    glEnableClientState(GL_VERTEX_ARRAY);
    glVertexPointer(3, GL_FLOAT, 0, corners);

    // Using DrawElements
    glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, indexes);

    glPopMatrix();

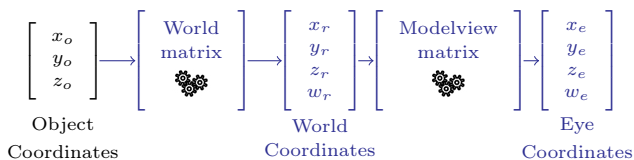
    // Swap buffers
    glutSwapBuffers();
}
```

Transformation Pipeline

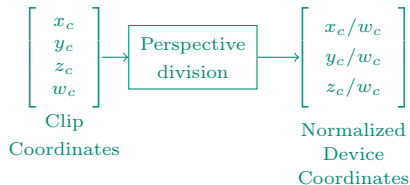
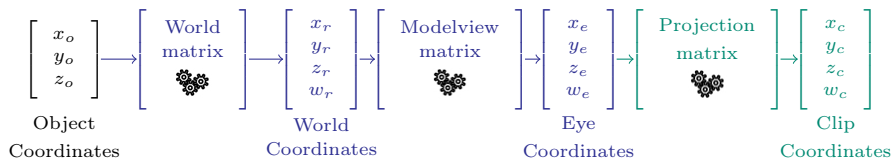
$$\begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix}$$

Object
Coordinates

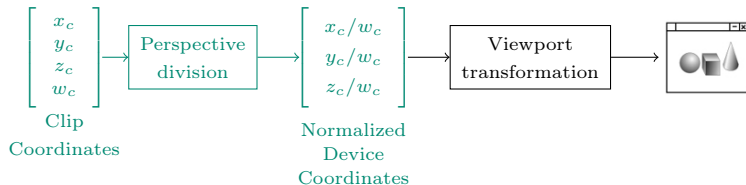
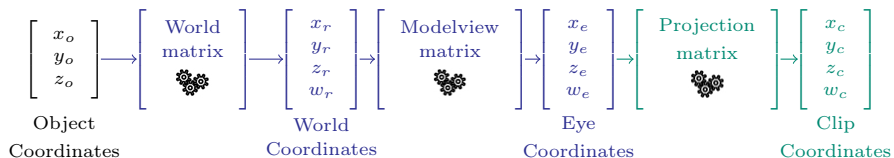
Transformation Pipeline



Transformation Pipeline



Transformation Pipeline



Matrices

Specifying Matrices

Defining a matrix:

```
GLfloat matrix[16];
```

$$\begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix}$$

Column-major matrix ordering

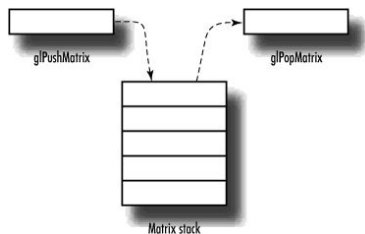
$$\begin{array}{cccc} \text{X axis direction} & \text{Y axis direction} & \text{Z axis direction} & \text{Translation/location} \\ \downarrow & \downarrow & \downarrow & \downarrow \\ \begin{bmatrix} X_x & Y_x & Z_x & T_x \\ X_y & Y_y & Z_y & T_y \\ X_z & Y_z & Z_z & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Position and orientation in 3D space

Matrix Stacks

OpenGL stores matrices in matrix stacks, one for each matrix mode
GL_MODELVIEW, GL_PROJECTION, or GL_TEXTURE

```
void glMatrixMode(GLenum mode)
```



Loading

Replacing the top of the stack with the matrix M :

```
void glLoadMatrix(const GLfloat* M)
```

Replacing the top of the stack with the identity matrix:

```
void glLoadIdentity()
```

Matrices

Postmultiplication

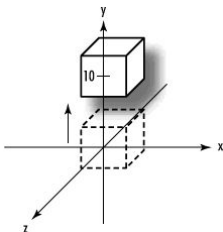
Replace the top of the stack T by the right multiplication TM where M is the input matrix:

```
void glMultMatrixf(const GLfloat* M)
```

Translation

Creates a matrix M that effects a translation, and postmultiplies M onto the top of the stack:

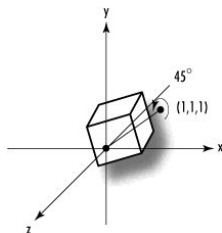
```
void glTranslatef(GLfloat x, GLfloat y, GLfloat z)
```



Rotation

Creates a matrix M that effects a rotation of $theta$ degrees around the line passing through $(0, 0, 0) : (x, y, z)$, and postmultiplies M onto the top of the stack:

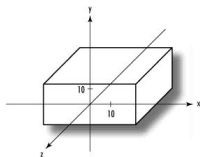
```
void glRotatef(GLfloat theta, GLfloat x, GLfloat y, GLfloat z)
```



Scaling

Creates a matrix M that effects a scale in the three axes, and postmultiplies M onto the top of the stack:

```
void glScalef(GLfloat x, GLfloat y, GLfloat z)
```



Modelview Transformation

View Transformation

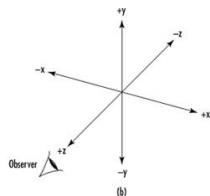
First transformation to be applied in the world coordinate scene. The view transform positions and orients the camera in the scene.

Model Transformation

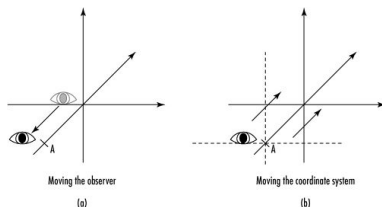
After placing the camera, model transformations move, rotate, and scale objects in the scene.

Modelview transformation

The modelview transformation is simply a global model transformation that is first applied to the entire scene to place the camera, followed by local model transformations that are applied to each dynamic object.



Camera after the view transform

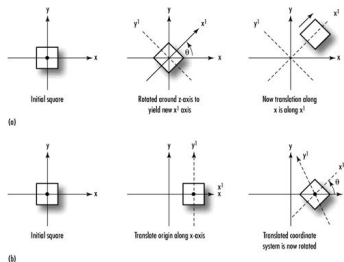


Modelview duality

Modelview Transformation

Modelview matrix

Each dynamic object in the scene is moved according to a 4×4 matrix VM , where V is the (global) view transform, and M is the (local) model transform of the object.

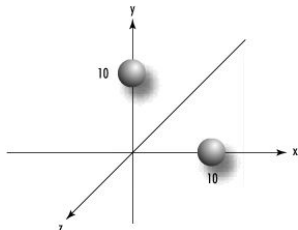


rotation/translation vs. translation/rotation

```
// Reset the modelview matrix
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

// Animate the first sphere
glPushMatrix();
glTranslatef(0.0f, 10.0f, 0.0f);
glutSolidSphere(1.0f, 15, 15);
glPopMatrix();

// Animate the second sphere
glPushMatrix();
glTranslatef(10.0f, 0.0f, 0.0f);
glutSolidSphere(1.0f, 15, 15);
glPopMatrix();
```



```

// Called to draw scene
void RenderScene(void)
{
    // Angle of revolution around the nucleus
    static GLfloat fElect1 = 0.0f;

    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Reset the modelview matrix
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    // Translate the whole scene out
    glTranslatef(0.0f, 0.0f, -100.0f);

    // Red Nucleus
    glColor3ub(255, 0, 0);
    glutSolidSphere(10.0f, 15, 15);

    // Yellow Electrons
    glColor3ub(255,255,0);

    // First Electron Orbit
    glPushMatrix();

    // Rotate by angle of revolution
    glRotatef(fElect1, 0.0f, 1.0f, 0.0f);

    // Translate out from origin to orbit distance
    glTranslatef(90.0f, 0.0f, 0.0f);

    // Draw the electron
    glutSolidSphere(6.0f, 15, 15);

    // Restore the viewing transformation
    glPopMatrix();

```

```

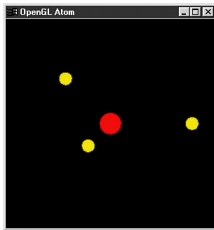
// Second Electron Orbit
glPushMatrix();
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glRotatef(fElect1, 0.0f, 1.0f, 0.0f);
glTranslatef(-70.0f, 0.0f, 0.0f);
glutSolidSphere(6.0f, 15, 15);
glPopMatrix();

// Third Electron Orbit
glPushMatrix();
glRotatef(360.0f, -45.0f, 0.0f, 0.0f, 1.0f);
glRotatef(fElect1, 0.0f, 1.0f, 0.0f);
glTranslatef(0.0f, 0.0f, 60.0f);
glutSolidSphere(6.0f, 15, 15);
glPopMatrix();

// Increment the angle of revolution
fElect1 += 10.0f;
if(fElect1 > 360.0f)
    fElect1 = 0.0f;

// Show the image
glutSwapBuffers();
}

```



Projection Transformation

Projection Matrix

The projection matrix defines the size and shape of the view volume. Thus, it determines the vertices that are visible to the observer.

The projection transformation multiplies eye coordinates by the projection matrix to produce clip coordinates.

Parallel Projection

```
void glOrtho(GLdouble left, right, bottom, top,
near, far)
```

Creates a parallel projection matrix and multiplies it on the top of the projection stack.

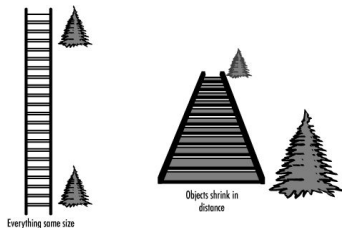
Perspective Projection

```
void gluPerspective(GLdouble fovy, aspect, near,
far)
```

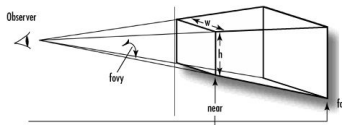
Creates a perspective projection matrix and multiplies it on the top of the projection stack.

- *fovy* is the field-of-view angle in the *y* direction.
- *aspect* is the ratio width/height.

the field-of-view angle in the *x* direction is *fovy* · *aspect*.



Parallel vs. Perspective



The Frustum

Projection Transformation

```
// Change viewing volume and viewport. Called when window is resized
void ChangeSize(GLsizei w, GLsizei h)
{
    GLfloat fAspect;

    // Prevent a divide by zero
    if(h == 0)
        h = 1;

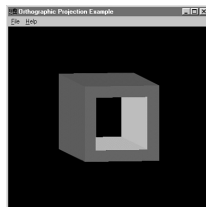
    // Set viewport to window dimensions
    glViewport(0, 0, w, h);

    fAspect = (GLfloat)w/(GLfloat)h;

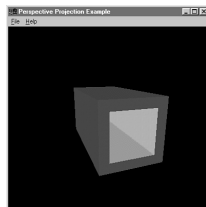
    // Reset coordinate system
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    // Produce the perspective projection
    gluPerspective(60.0f, fAspect, 1.0, 400.0);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```



Tube in parallel projection



Tube in perspective projection

Projection Transformation

```
// Change viewing volume and viewport. Called when window is resized
void ChangeSize(GLsizei w, GLsizei h)
{
    GLfloat fAspect;

    // Prevent a divide by zero
    if(h == 0)
        h = 1;

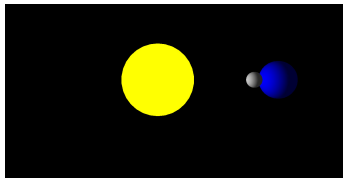
    // Set viewport to window dimensions
    glViewport(0, 0, w, h);

    // Calculate aspect ratio of the window
    fAspect = (GLfloat)w/(GLfloat)h;

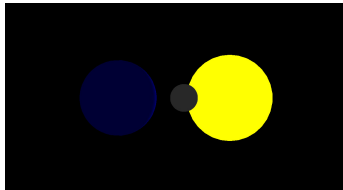
    // Set the perspective coordinate system
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    // Field of view of 45 degrees, near and far planes 1.0 and 425
    gluPerspective(45.0f, fAspect, 1.0, 425.0);

    // Modelview matrix reset
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```



Earth on near side of the sun



Earth on far side of the sun

Projection Transformation

```
// Called to draw scene
void RenderScene(void)
{
    // Earth and moon angle of revolution
    static float fMoonRot = 0.0f;
    static float fEarthRot = 0.0f;

    // Clear the window with current clearing color
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Save the matrix state and do the rotations
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();

    // Translate the whole scene out and into view
    glTranslatef(0.0f, 0.0f, -300.0f);

    // Set material color, to yellow
    // Sun
    glColor3ub(255, 255, 0);
    glDisable(GL_LIGHTING);
    glutSolidSphere(15.0f, 15, 15);
    glEnable(GL_LIGHTING);
```

```
// Position the light after we draw the Sun!
glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
```

```
// Rotate coordinate system
glRotatef(fEarthRot, 0.0f, 1.0f, 0.0f);
```

```
// Draw the earth
glColor3ub(0,0,255);
glTranslatef(105.0f,0.0f,0.0f);
glutSolidSphere(15.0f, 15, 15);
```

```
// Rotate from Earth-based coordinates and draw moon
glColor3ub(200,200,200);
glRotatef(fMoonRot,0.0f, 1.0f, 0.0f);
glTranslatef(30.0f, 0.0f, 0.0f);
fMoonRot+= 15.0f;
if(fMoonRot > 360.0f)
    fMoonRot = 0.0f;
```

```
glutSolidSphere(6.0f, 15, 15);
```

```
// Restore the matrix state
glPopMatrix(); // Modelview matrix
```

```
// Step Earth orbit 5 degrees
fEarthRot += 5.0f;
if(fEarthRot > 360.0f)
    fEarthRot = 0.0f;
```

```
// Show the image
glutSwapBuffers();
}
```