

# Introduction to OpenGL and GLUT

January 14<sup>th</sup> and 16<sup>th</sup> 2009

MAE 574, Virtual Reality Applications

Instructor: Govindarajan Srimathveeravalli

# Instructor Information

- Instructors :
  - Govindarajan Srimathveeravalli (gks2 [AT] buffalo [dot] edu).
  - Dr T Kesavadas ([kesh@eng.buffalo.edu](mailto:kesh@eng.buffalo.edu))
- Old course website (from 2007)
  - <http://wings.buffalo.edu/courses/sp07/mae/574-410/>
- Office:
  - 328 Jarvis Hall
- Hours:
  - Tuesday and Thursday, 3-5 PM
- Grading (tentative):
  - Homework : 30%
  - Mini projects (2) : 30%
  - Final project : 25%
  - Midterm exam : 10%
  - Instructor discretion : 5%

# Prerequisites and Resources

- First resource
  - OpenGL Programming Guide: The Official Guide to Learning OpenGL (aka) “*The Red Book*”
  - Available online, ‘*Google it*’ or buy it from Amazon
  - Computer Graphics Using OpenGL – F.S Hill (recommended, but not required)
- Other resources
  - [http://oss.sgi.com/cgi-bin/cvsweb.cgi/projects/ogl-sample/main/gfx/samples/redbook\\_2ed/](http://oss.sgi.com/cgi-bin/cvsweb.cgi/projects/ogl-sample/main/gfx/samples/redbook_2ed/)
  - NeHe <http://nehe.gamedev.net/>
  - Others include [Gamasutra](#), [opengl.org](http://opengl.org), [gamedev](#), [flipcode](#), [gametutorials](#), [lighthouse3D](#), various university courses, etc.

# Required

- A working knowledge of C++
  - Thinking in C++ (Bruce Eckel)  
<http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html>
  - Robert Lafore or Dietel & Dietel series.
- Visual Studio 2005 or 8.0 (.Net not recommended)
- Available in 1019 Furnas and UBMicro, Microsoft – UB tie up in Bell.
- OpenGL (comes with install of the IDE) and Glut (Needs to be installed separately) libraries
- MESA etc. ?

# Syllabus

- **OpenGL and graphics**
- **Input devices**
- **Display systems**
- **Computational geometry**
- **Collision detection**
- **Physics**
- **Additional topics**

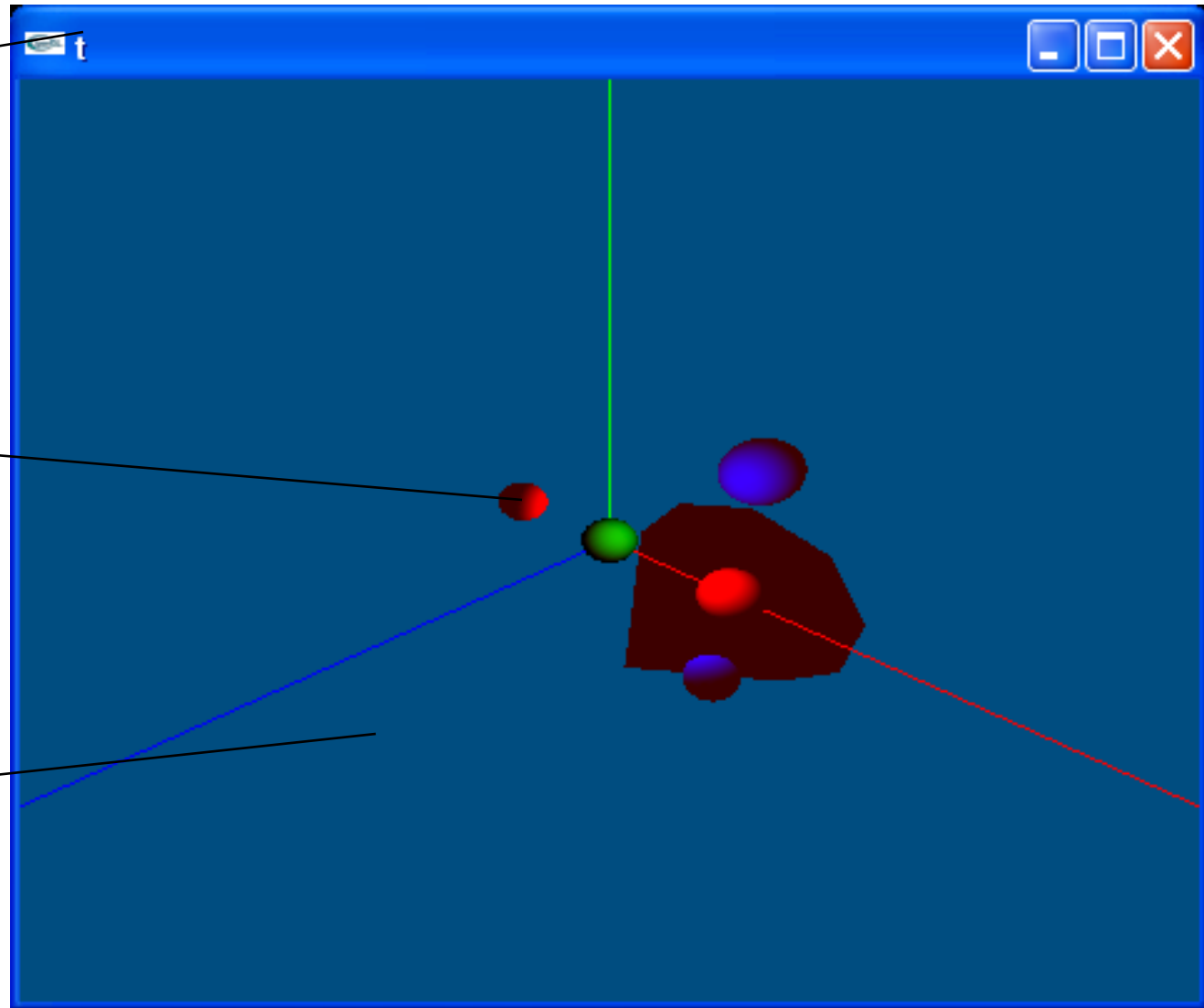
# “Hello World” in OpenGL + Glut

Window/Event Handling (Glut)

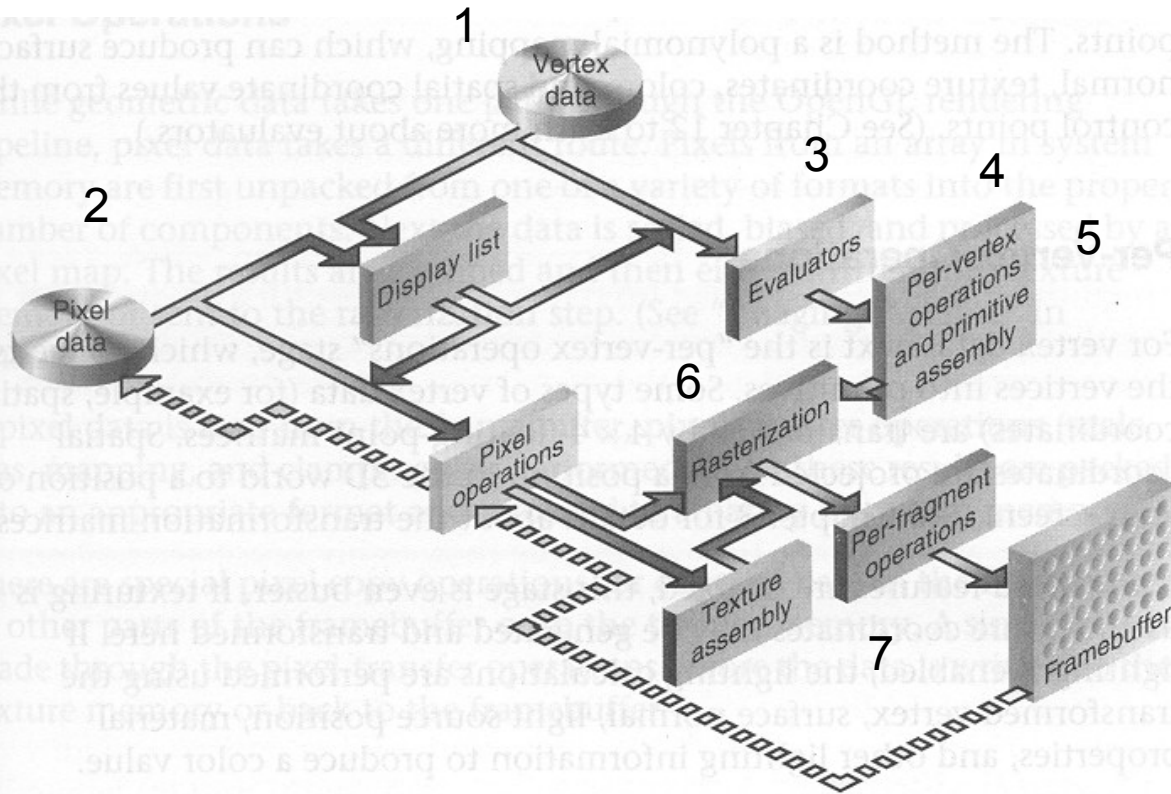
Graphical Objects (OpenGL)

Rendering (OpenGL)

Class 1&2

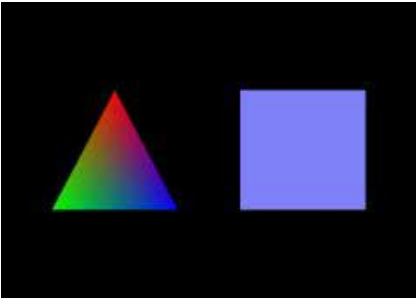


# Role of OpenGL



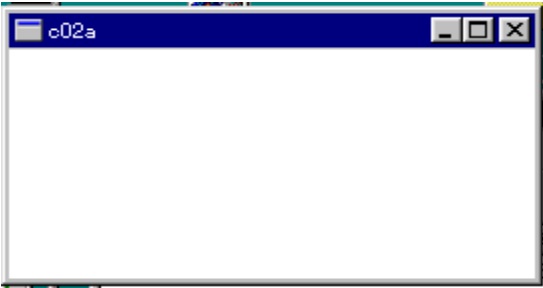
1. Geometric Data
2. Pixel Data (bitmaps, textures etc.)
3. Evaluators to determine values for parametric curves
4. Vertex data needs to be converted to primitives
5. Primitive opns :- Culling, clipping, other imp calculations
6. Rasterization :- Conversion of geometric and pixel data into framebuffer (pixel) based information
7. Per fragment operations:- Where framebuffer information is modified, like fog, antialiasing etc.

# Role of Glut ?



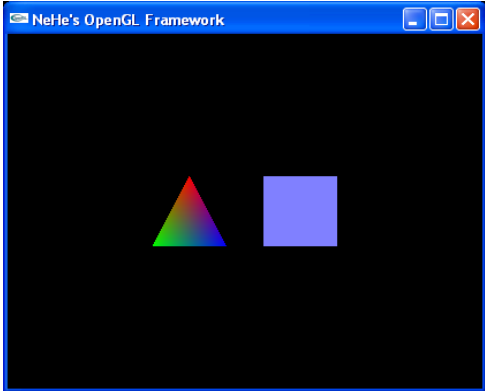
Rendering (OpenGL)

+



Window + Frame  
(Glut)

=



(Code courtesy NeHe)

+

Your Callback Functions



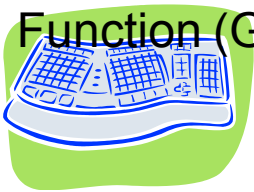
Generic Controller  
Function (Glut)



Event (Clicking Moving Typing)



Generic Keyboard  
Function (Glut)



Generic Mouse Function  
(Glut)



# GLUT

- Trivia :- Developed by Mark Kilgard and Nate Robins
- Out of date, yet convenient. (Last version in '01)
- Available from <http://www.opengl.org/resources/libraries/glut/> or <http://freeglut.sourceforge.net/>
- Move to Windows – NeHe arrangement ?
- What comes in the package?
  - glut32.lib, glut32.dll and glut32.h

# Glut Setup Basics

- If you have “admin” access to a machine and will use it primarily
  - Unzip (compile if necessary) the package from either Nate Robins or from freeglut
  - Place the glut.dll file into your systems directory i.e. WINDOWS\system32
  - Create a “permanent” folder for your glut.lib and glut.h files, lets say in C:\GLUT
  - Fire up Visual Studio and do once.
    - Go to Tools ->Options->Projects and Solutions->VC++ Directories
    - Choose Include from the drop down list and point it to the newly created GLUT folder
    - Rinse and repeat for libraries
- If you do not have “admin” on your machine
  - Place all the files (.dll, lib and .h) into a suitable “GLUT” directory.
  - Do step four from previous list
- Also, right click on my computer, click properties->advanced->environment variables
  - Here click new, add GLUT as variable and the path to the “GLUT directory
- Now you are all set to start programming using Glut

# Various Pieces

- gl: The basic libraries.
  - Data management, geometry, properties etc.
- glu: Core implementation of OpenGL
  - Math extensions for shapes and curves, projections and viewing
- glut: Generic platform independent library.
  - Windowing and UI functions
- Others include glSL (glShading language), gl\_ARB (Architecture review board extensions)
- glX/wgl/pgl : Windowing system for X windows, Win 98/NT/XP systems and Mac/OS X.
  - Not relevant to our purposes

# “Hello World” in OpenGL

Create and initialize a window  
(Glut)



Create and define graphical  
objects (OpenGL)



Draw objects onto screen  
(Glut+OpenGL)

A painter, frame and  
canvas analogy!



# Setting up the “first” window

- Window management
  - glutInit( int\*, char\*\* )
    - Initialize glut and the window, accepts “*command line arguments*”
  - glutInitDisplayMode( int )
    - Determine the type of buffering (Single or Double), drawing color, display parameters etc.
  - glutInitWindowPosition( int, int )
    - Where is the window going to sit initially
  - glutInitWindowSize( int, int )
    - How big is the window going to be
  - glutCreateWindow( char\* )
    - Create the actual window
  - glutMainLoop( )
    - Go into an infinite loop of drawing and accepting event input into the window
- Additional
  - glutDisplayFunc( ) [ Callback Function? ]
    - Where all the “*drawing*” code is at

# Callback Function ?

- Generic function which calls your “custom” function at occurrence of an “event”
  - whenClickMouse( void (\*DoSomething()) )
- Don’t exactly pass a function, but a function pointer
- Opportunities & Pitfalls
  - Cannot pass external arguments
  - Necessity for global variables
  - Big headache when implemented as OOP

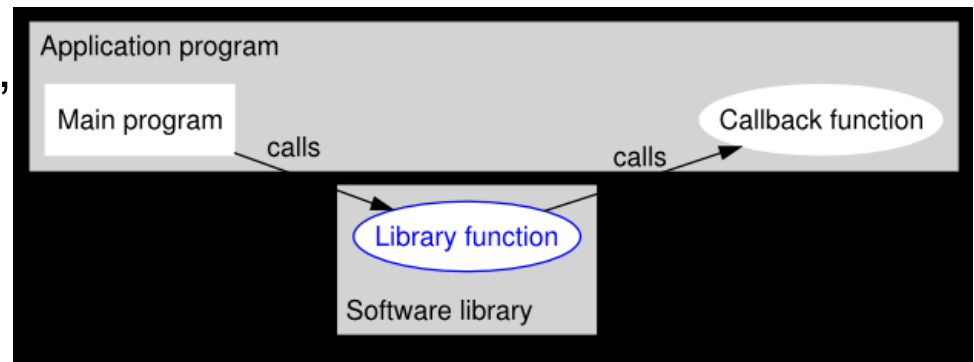


Image courtesy Wikipedia

# Other Important Commands

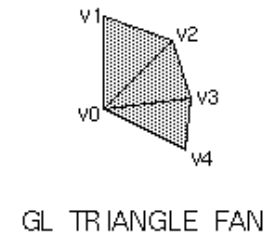
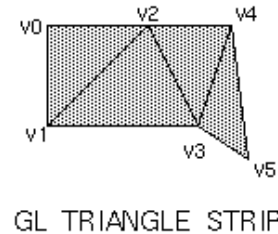
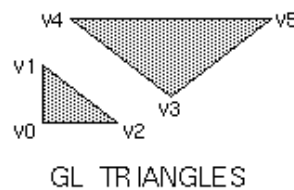
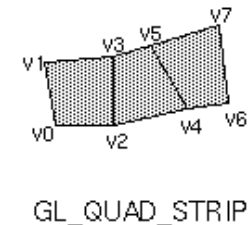
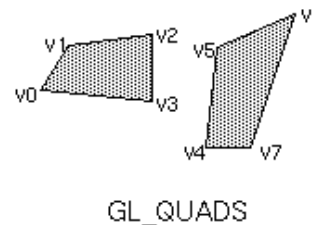
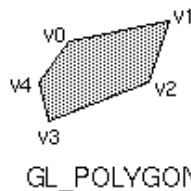
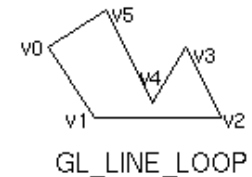
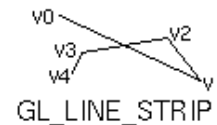
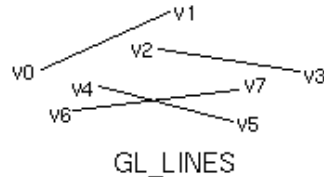
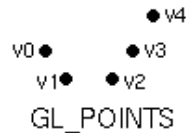
- `glutReshapeFunc( (*func) , int w, int h )`
  - Makes sure the graphical objects are drawn correctly when windows resizes etc.
  - What about non square windows ?
- `glutIdleFunc( (*func) )`
  - Code that is executed when window is not processing any “events”
- `glutKeyboardFunc( (*func), unsigned char, int, int )`
  - Accept keyboard inputs
- `glutMouseFunc( (*func), int, int , int )`
  - Accept mouse click inputs
- `glutMotionFunc( (*func), int, int )`
  - Handle mouse “drag” inputs

# Collected Commands

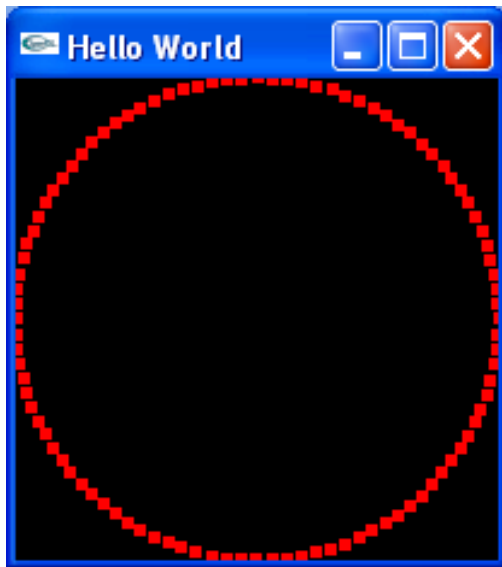
- glClear
- glColor
- glBegin() – glEnd()
  - glVertex
- glFlush()
- glClearColor()
- glMatrixMode()
- glOrtho
- glutPostRedisplay()

# Primitives

- Points (Vertices)
- Lines
- Line strip
- Line loop
- Triangles
- Triangle Strip
- Quadrilaterals (Quads)
- Quad Strips
- Polygons



# Draw Some More Primitives



# More on Primitives

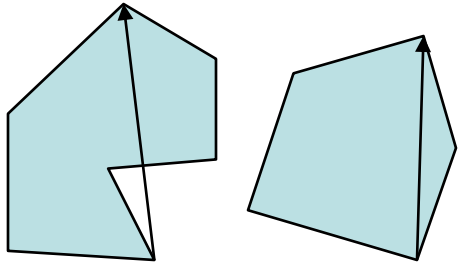
- Points
  - `glPointSize()` :Sets the size of the point, specified in pixel widths
- Lines
  - `glLineWidth()`
  - `glLineStipple()` : Takes in two values and draws line in “*patterns*”
- Polygons
  - Convexity (line joining any two vertex does not intersect an edge)
  - Tessellation (convert polygon into triangles)
    - Any three points lie on a plane
    - Problem with 3D twist and shear
  - Front and back faces of a polygon
    - f.e. `glPolygonMode( GL_FRONT, GL_FILL );`
  - Ordering of vertices
    - Counterclockwise – front facing, but can be changed using `glFrontFace()`
    - Hide a face using `glCullFace()`

- Normalization
  - Why?
- How do you know if 3 points lie on a plane?
  - Why is this important?
- Ordering of polygons
  - What is front facing?
  - Vector closure
  - Dot product
- How to find convexity of polygons?

# Associated Stuff

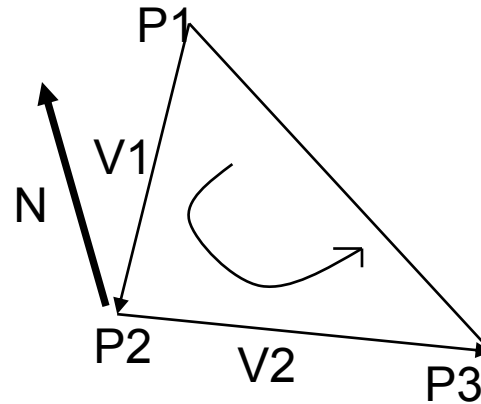
- All come between glBegin and glEnd
- Define properties of the primitive
- glVertex
  - Basic point definition
- glColor
  - Color definition, can be per-vertex!
- glNormal
  - Normal to given a plane
  - Used for a variety of calculations, shading, lighting etc.
  - How do you calculate the normal ?
    - Head – Tail on a triangle
    - Determine 2 vectors like that and find their cross product using the “matrix form”
- Others include
  - glMaterial
  - glTexCoord
  - glIndex

# Convexity, Ordering and Normals



Non-convex

Convex



- Always order your vertices in counter clockwise direction

- Normals are to be calculated and included for each polygon

- Calculate vectors from vertices as Head-Tail, f.e.  $V1 = P2 - P1$

- Need two vectors on a plane to get normal to it

Convexity: Line joining two vertices should not intersect any edge

Convexity useful for number of operations, including correct rendering, shading, collision detection

$$\text{Cross Product } N = V1 \times V2 = (V1y \cdot V2z - V1z \cdot V2y) + (V1x \cdot V2z - V1z \cdot V2x) + (V1x \cdot V2y - V1y \cdot V2x)$$

	X	Y	Z
	V1x	V1y	V1z
	V2x	V2y	V2z

- Vectors calculated should be “*head to tail*”

- The “*right hand rule*” can be applied to determine the normal (thick arrow pointing out of the screen for the triangle)

- Calculate normal using determinant or cross product

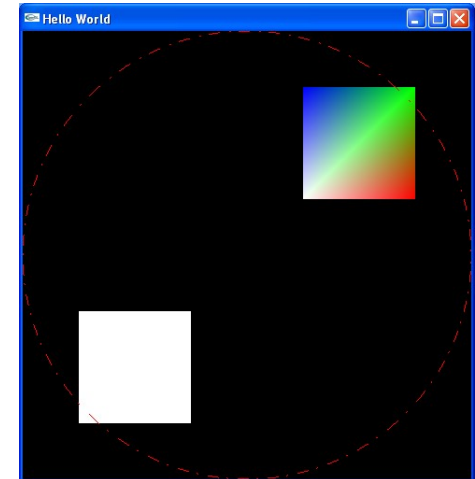
# Coloring Your Objects

- Colors can either be from [0 – 1] or [0 – 255], we stick to former
- So typically all we need is glColor3f( R G B)
- What about Alpha ?
  - As long as we stick to just glColor(), alpha is not needed.
  - Will come into play when using materials, (later)
- Looking back at the polygon example
  - The first polygon is white in color, the second; each vertex has its own color
  - Color of the polygon is taken from the immediately preceding “state”
  - State Management
    - Number of things can affect the way primitives are rendered (graphical objects like lines, polygons etc.)
    - Things that affect them include color, lighting , texture etc. are called *states* or *state variables*
    - States can be turned “on” or “off” using glEnable and glDisable
    - Once a state is enabled it will affect all primitives succeeding it in code till it is disabled or another state is enabled which overrides the previous one.
    - F.e. color of polygon changing per object and per vertex.

```
// Coloring
glColor3f( 1., 1., 1. );

// Matrix operation for rotation
glPushMatrix();
// For rotation
//glRotatef( spin, 0., 0., 1. );
// Actual code to do the drawing
glBegin( GL_POLYGON );
    glVertex3f( 0.25, 0.25, 0. );
    glVertex3f( 0.75, 0.25, 0. );
    glVertex3f( 0.75, 0.75, 0. );
    glVertex3f( 0.25, 0.75, 0. );
glEnd();
glPopMatrix( );

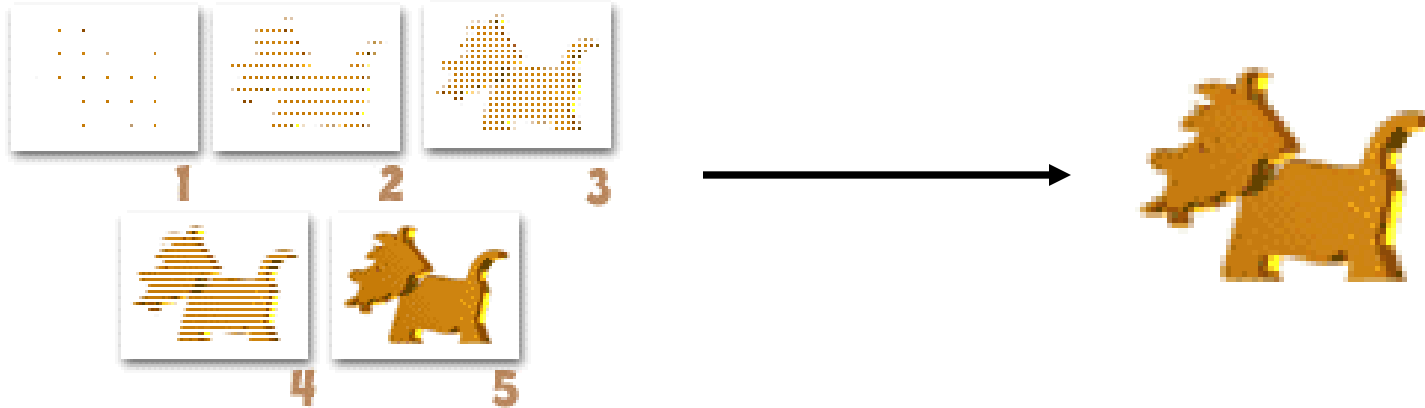
//with colors
glBegin( GL_POLYGON );
    glColor3f( 1., 1., 1. );
    glVertex3f( 0.25, 0.25, 0. );
    glColor3f( 1., 0., 0. );
    glVertex3f( 0.75, 0.25, 0. );
    glColor3f( 0., 1., 0. );
    glVertex3f( 0.75, 0.75, 0. );
    glColor3f( 0., 0., 1. );
    glVertex3f( 0.25, 0.75, 0. );
glEnd();
```



# Going back ...

- Clearing stuff up before drawing
  - Background
  - Depth
  - Others, including (accumulation, stencil etc. To be covered later)
- Need to set values before clearing
  - i.e. `glClearColor( RGBA )`
- Recap :- Main components of a “rendered scene”
  - Object (done)
  - Color (done)
  - Movement (next)
  - Light
  - Texture

# Basic Animation



Draw – Erase – Redraw , The way the computers do it !  
Computationally more cheaper than updating changed pixels



# Basic Animation

- Sequence of pictures at 24 frames per second
- Monitor refresh rate – 60 fps
- But also account for redraw and clearing times
- Double buffering, i.e. two frame buffers
  - One is drawn, the other displayed
  - Draw – Erase – Swap, repeat
- Difference between Single and Double Buffering
  - One artist with one sketchpad versus two artists with sketchpads working in tandem, one preparing the current drawing and the other preparing the next one.
- OpenGL does not inherently support double buffering
- Glut comes in here
  - `glutSwapBuffers()` , Forces the swapping of the two display buffers.
  - To be put at the end of the “display” code.
  - `GLUT_DOUBLE`, initializes the window to support double buffering

# Homework 1 ☹️ - Due:Jan 23<sup>rd</sup>

- Form groups of two!! Not three, not one, but two.
- Create a glut window
  - Call it homework 1
  - Parameters for window size and position must be fed in through “command line”
- Create functions that will draw lines, triangles, rectangles and circles (ellipses are a bonus)
- Use key board input to choose shape to be draw (l, t, r, c and e?)
- Pop up polygons when using left mouse clicks
  - The colors and the sizes of polygons popped up must be “random”
  - The linewidth and “stippling” used for the lines must be “random”
- Draw lines based on data from left mouse clicks and completion on right mouse click
- Have an animation mode
  - Have two forms of animation, canned and real time
  - In real time animation mode if mouse is clicked near existing objects they should “scatter or run away”
    - Use a math model (what would be the equation like?) to perform scattering
    - Think in terms of a water droplet on a surface, nearer particles are affected more than farther particles
  - Another form in the real time animation mode could be to make the on screen objects follow the mouse pointer
    - Can this be the inverse problem of the previous case?
  - A canned animation mode would cause the press of a button to make your objects to start moving in “random” directions with “random” velocities
    - But the objects should never leave your “screen window”
- Note: These requirements for objects on screen (color, size etc.) and movement, will require you to create a “structure” for each of these objects.
- You would also need to maintain an “array” of these objects and you will need to know their “position” with respect to the screen