CS3241 Computer Graphics

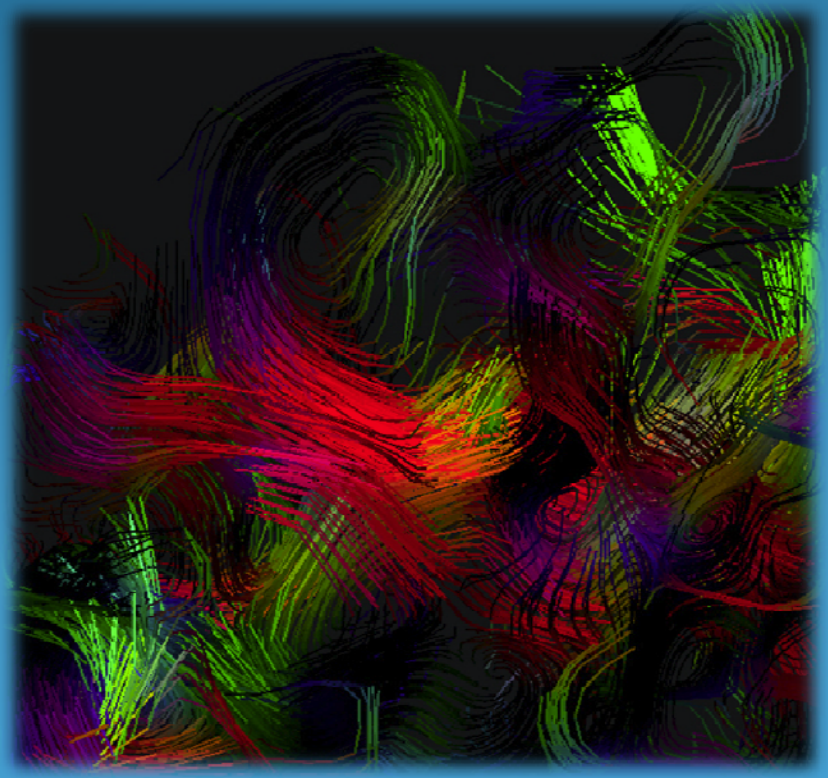
Remember there is a Facebook Page

https://www.facebook.com/NUSCG

C++

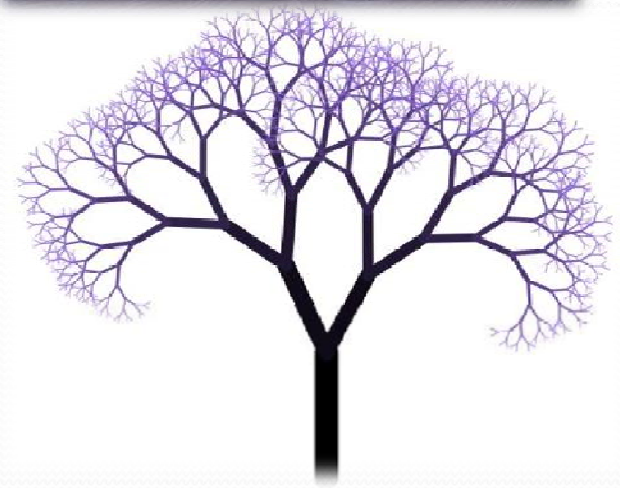
- Download “Microsoft Visual Studio Professional” 2008/2010/2012
  - Not Express!
  - You may need an SOC account if you are not from SOC
    - Please get an account from the helpdesk at COM1
- Please do it this week
- `http://msdn70.e-academy.com/elms/Storefront/Home.aspx?campus=socnusingapore_cs`

2D Graphics and OpenGL Primer



Goal of Today

- Everyone can draw some profound pictures in 2D

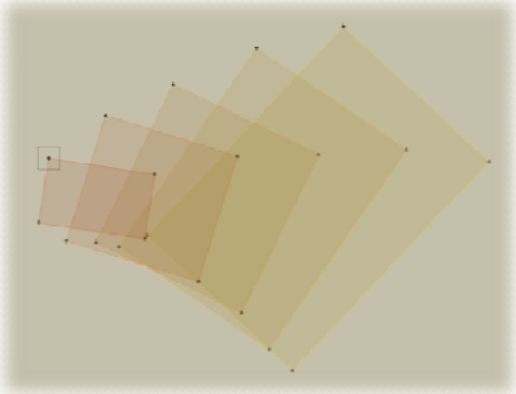


What you can do after today

- Games and animations in 2D
  - *With OpenGL, GLUT and C++*

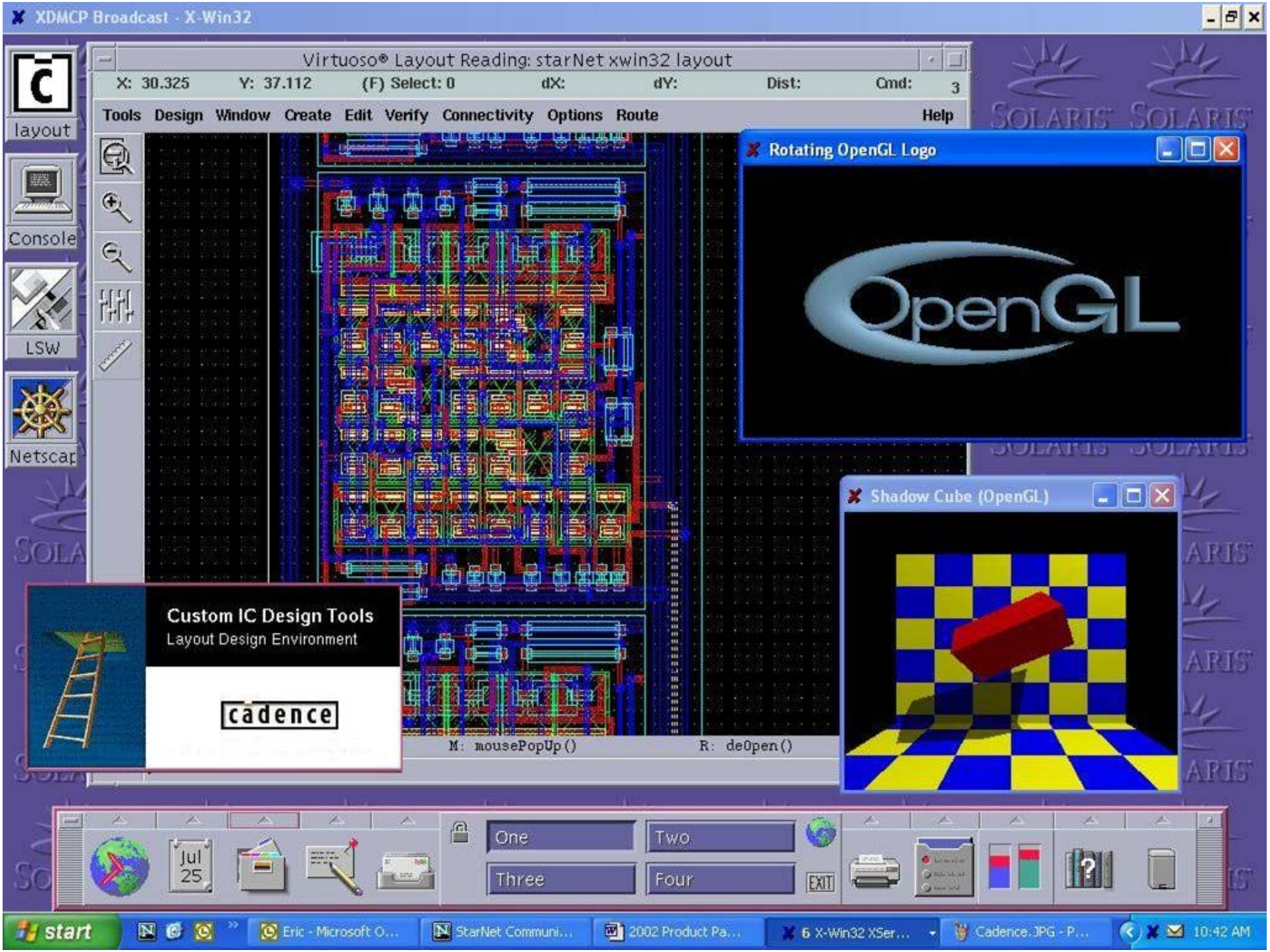
Approach

- Not top-down
  - Abstract/overall > details
- Bottoms-up !!
  - Intuition/motivation first
  - Teach-on-demand approach
    - Problem driven,
    - or I will rather say motivation driven
- Be arty-farty



OpenGL and GLUT

- In the old days, the first library for programming window system is called
  - X-windows
  - You need to program two pages of code to open 1 single window…. with nothing.
- **OpenGL**
  - From the GL library (1992) from Silicon Graphics
  - Handle graphics routines and window events handlings
- **GLUT** - The OpenGL Utility Toolkit
  - pronounced like the "glut" in gluttony
- In a word, save you a lot of trouble by
  - API: Application Programming Interface


XDMCP Broadcast - X-Win32
Virtuoso® Layout Reading: starNet xwin32 layout
X: 30.325
Y: 37.112
(F) Select: 0
dX:
dY:
Dist:
Cmd:
3
Tools
Design
Window
Create
Edit
Verify
Connectivity
Options
Route
Help
layout
Console
LSW
Netscap
Rotating OpenGL Logo
OpenGL
Shadow Cube (OpenGL)
Custom IC Design Tools
Layout Design Environment
cadence
M: mousePopUp()
R: deOpen()
One
Two
Three
Four
EXIT
Jul
25
start
Eric - Microsoft O...
StarNet Communi...
2002 Product Pa...
6 X-Win32 XSer...
Cadence.JPG - P...
10:42 AM

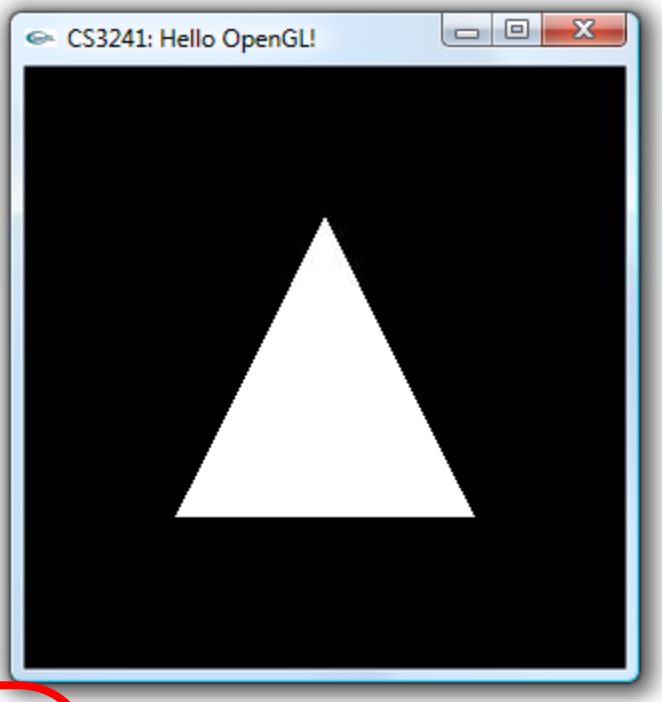
Setup GLUT Library

- Download GLUT (from the link in the course website)
- If you are using MS Windows, follow the README file
- Copy the files:
  - `glut32.dll` to `%WinDir%\System`,
  - `glut32.lib` to `%your vc directory%\lib`, and
  - `glut.h` to `%your vc directory%\include\GL`
- DONE!

First OpenGL Program

```
#include <GL/glut.h>

void mydisplay(){
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
        glVertex2f(-0.5, -0.5);
        glVertex2f(0, 0.5);
        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
```

Depends on where do you put your glut.h

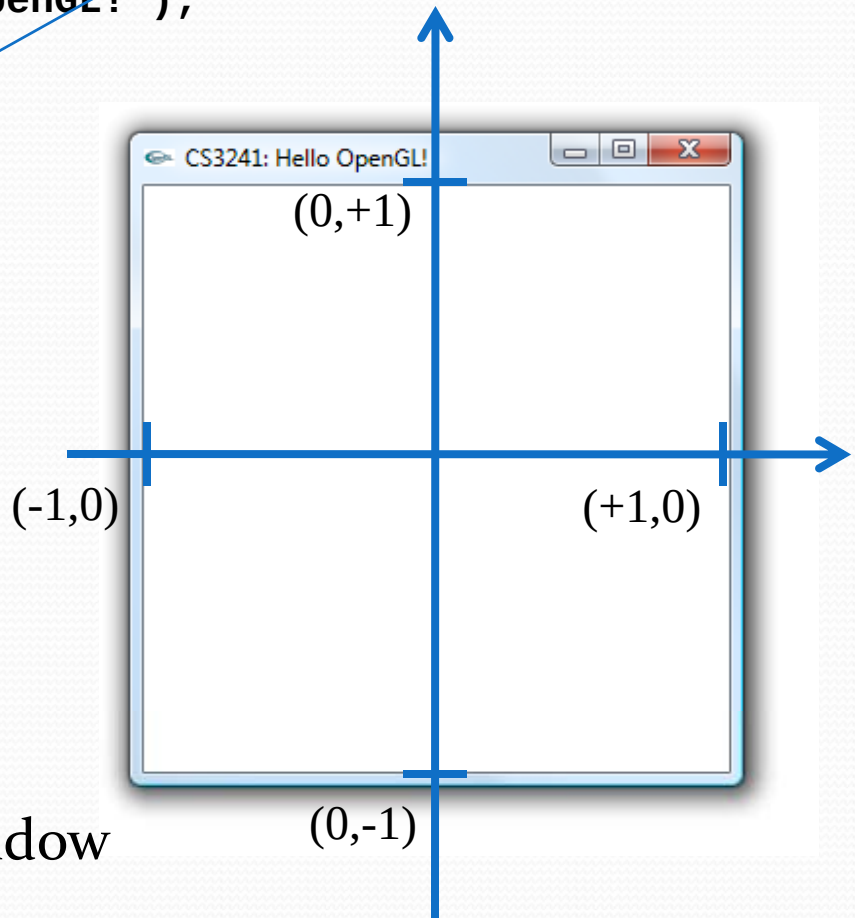



```
int main()
{
    glutCreateWindow("CS3241: Hello OpenGL!");
    glutDisplayFunc(mydisplay);
    glutMainLoop();
}
```

First OpenGL Program

```
int main()
{
    glutCreateWindow("CS3241: Hello OpenGL!");
        // simply create a window
        // with title

    glutDisplayFunc(mydisplay);
        // "register" the function
        // "mydisplay" for display

    glutMainLoop();
        // starts the window loop
}
```

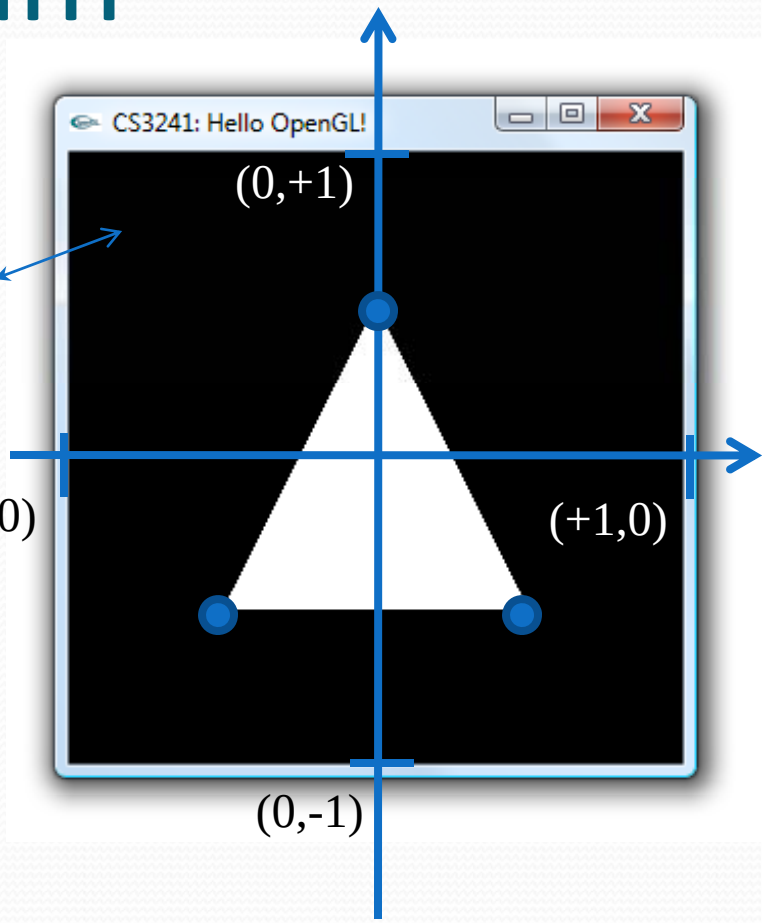
May sound weird to you: passing a "function" as a parameter



- Default window "dimension":
  - (-1,-1) to (+1,+1)
  - Not the "actual size" of the window

First OpenGL Program

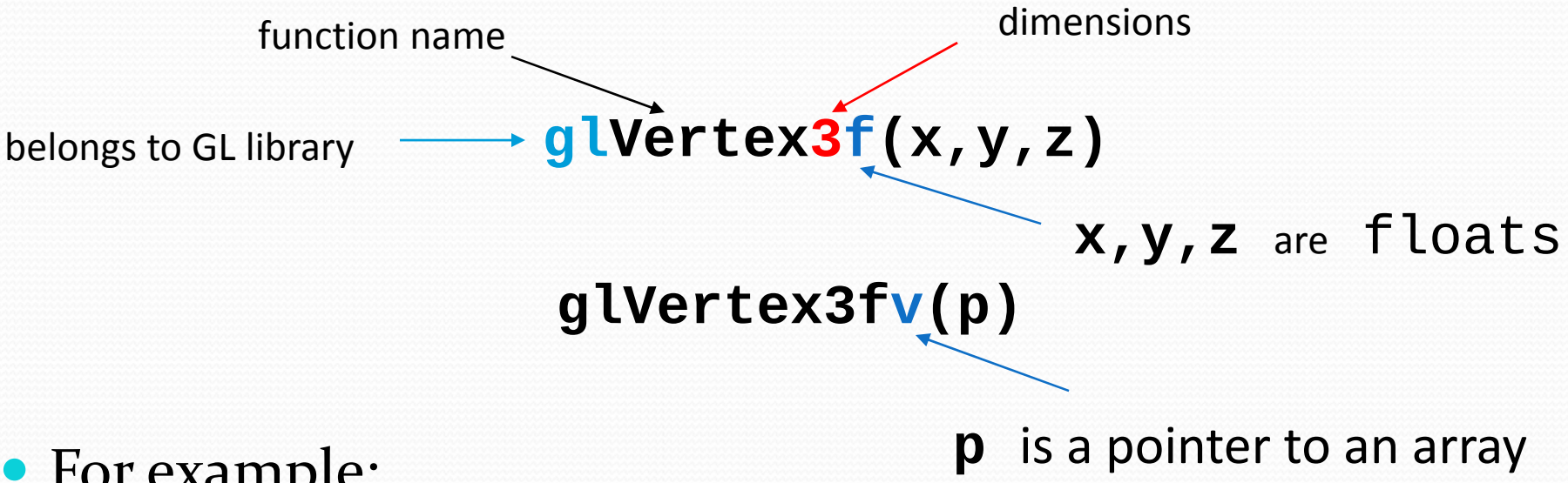
```
#include <GL/glut.h>

void mydisplay(){
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
        glVertex2f(-0.5, -0.5);
        glVertex2f(0, 0.5);
        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
```



Clear a kind of buffer
(now it is the color buffer)
("buffer" means memory space)

Note: the default colors for background and drawing are black and white respectively

OpenGL Function Format



- For example:

```
GLfloat p[2] = {-0.5,0.5};
glBegin(GL_POLYGON);
  glvertex2fv(p);
  …
```

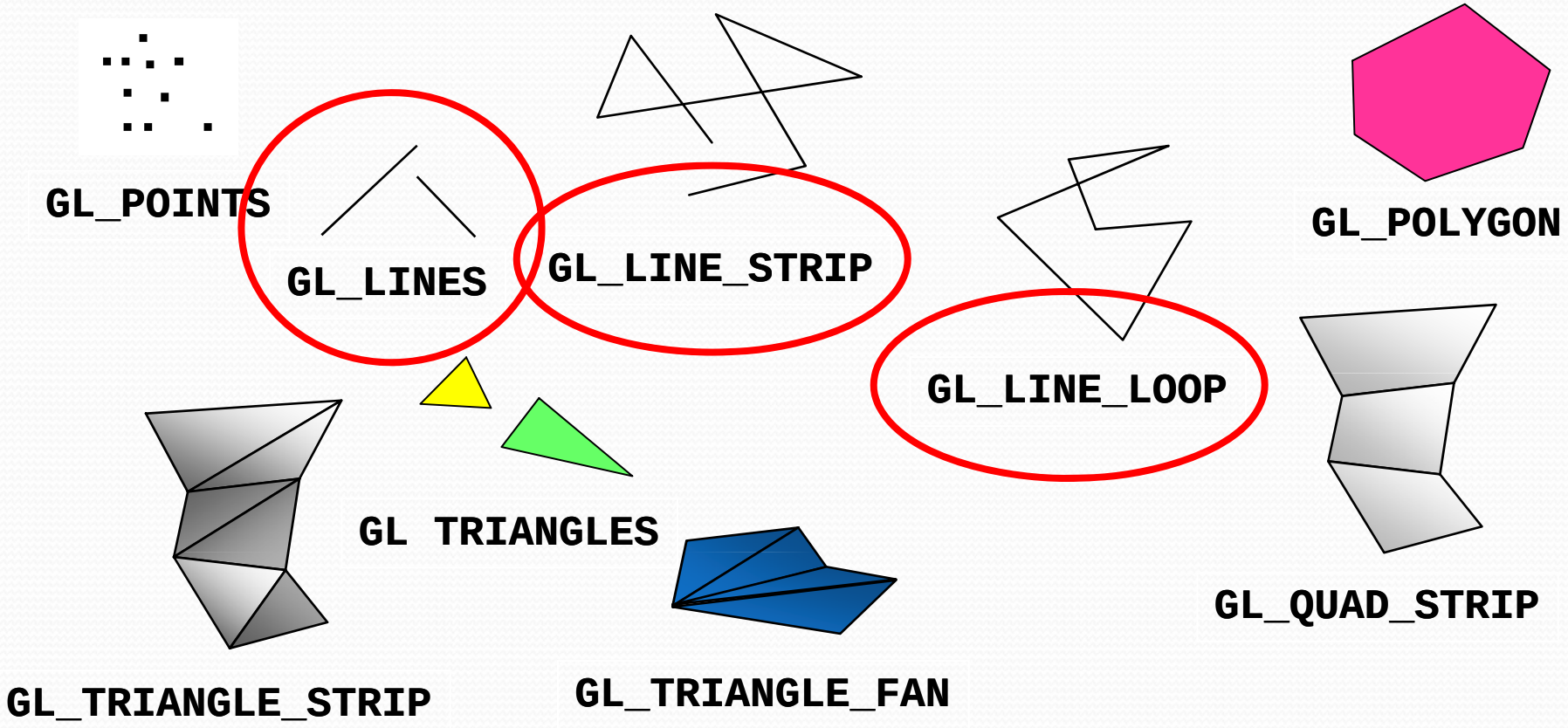
Usage for `glBegin()` and `glEnd()` pair

```
glBegin(GL_POLYGON);
        glVertex2f(-0.5, -0.5);
        glVertex2f(0, 0.5);
        glVertex2f(0.5, -0.5);
glEnd();
```

Begin to draw some **primitives**, e.g. polygon, lines, points, etc.

A list of commands, e.g. vertex, normal, texture, etc.

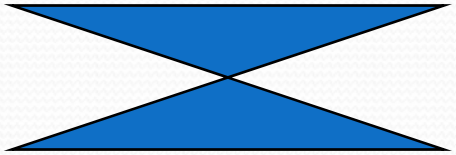
OpenGL Primitives for `glBegin(?)`



See: http://www.opengl.org/sdk/docs/man/ [click **`glBegin()`**]

Polygon Issues

- OpenGL will only display polygons **correctly** if they are
  - Simple:
    - Edges cannot cross within a polygon
  - Convex:
    - All interior angles are less than 180 degree
  - Flat:
    - All vertices are in the same plane (in 3D)
- OpenGL **may or may not** produce desired output if these conditions are violated
- The best is to use triangles



non-simple polygon

non-convex polygon

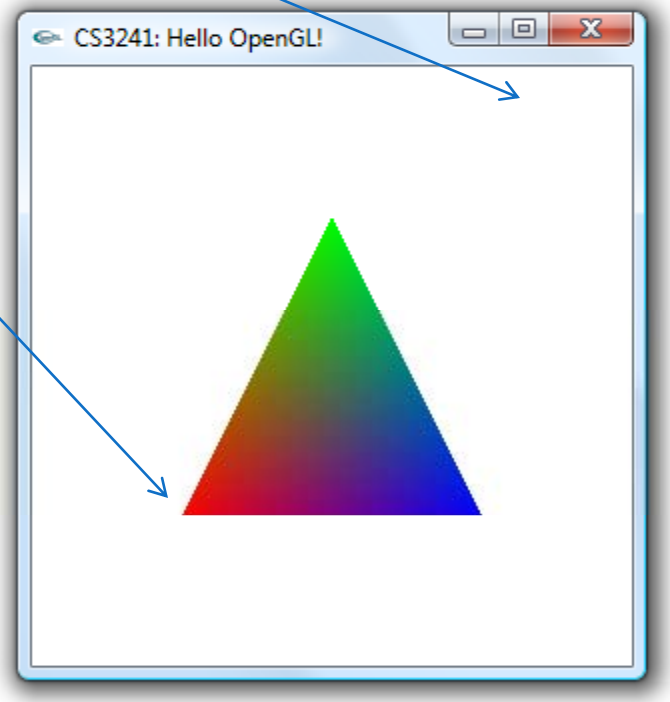
Changing Colors

```
void mydisplay(){
    glClearColor(1.0,1.0,1.0,1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
       glColor3f(1.0,0.0,0.0);
       glVertex2f(-0.5, -0.5);
       glColor3f(0.0,1.0,0.0);
       glVertex2f(0, 0.5);
       glColor3f(0.0,0.0,1.0);
       glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
```

Set Clear color to be white
(But **nothing changed** until
`glClear` is called)

Set this vertex to be red

(red, green, blue)



2D Transformation

Here comes the mathematics

2D Transformation

- Vectors
- Reference Frame
- Transformation
  - Translation
  - Rotation
  - Scaling

Vector/Matrix Representations

- A vector (x,y) can be represented by a column or row **matrices**:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{ or } \begin{bmatrix} x & y & 1 \end{bmatrix}$$

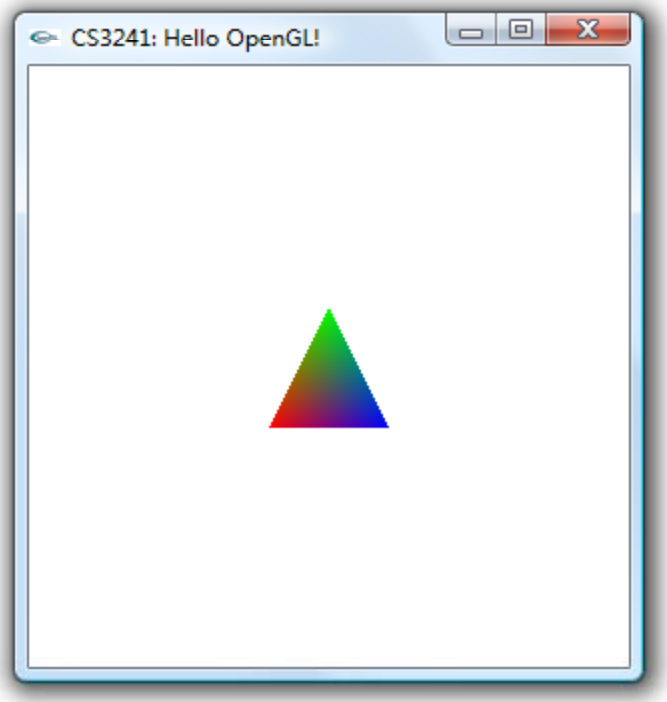
- Which is called **Homogeneous coordinates**
- What if the last coordinate is not 1?

$$\begin{bmatrix} x & y & w \end{bmatrix} = (\frac{x}{w}, \frac{y}{w})$$

- If w=0, the point is at infinity in the direction (x,y)
  - E.g. (0,1,0) is the point at the "north" most
- In this course, we will use the column matrix representation

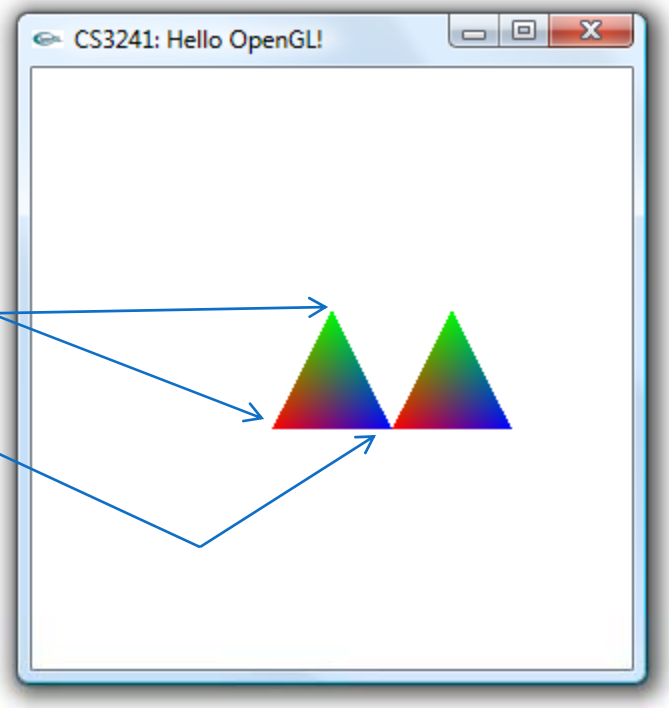
Let's Start From This

```
void drawASmallTri()
{
    glBegin(GL_POLYGON);
       glColor3f(1.0,0.0,0.0);
       glVertex2f(-0.2, -0.2);
       glColor3f(0.0,1.0,0.0);
       glVertex2f(0, 0.2);
       glColor3f(0.0,0.0,1.0);
       glVertex2f(0.2, -0.2);
    glEnd();
}
void mydisplay(){
    glClearColor(1.0,1.0,1.0,1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    drawASmallTri();
    glFlush();
}
```



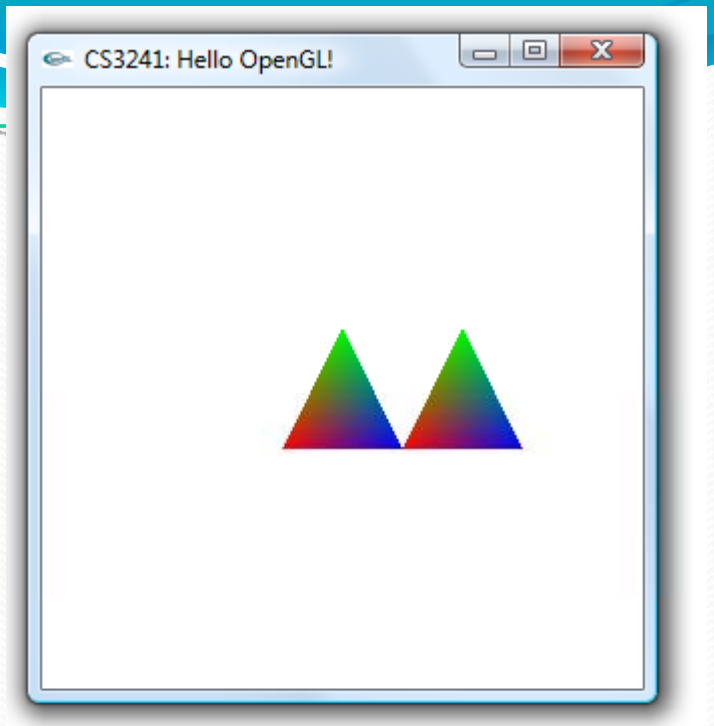
How Do I Draw This?

- The left triangle is the original one
  - **a**: (-0.2,-0.2)
  - **b**: (0.2,-0.2)
  - **c**: (0,0.2)
- The right one is
  - (0.2,-0.2) = **a** + (0.4,0)
  - (0.6,-0.2) = **b** + (0.4,0)
  - (0.4,0.2) = **c** + (0.4,0)



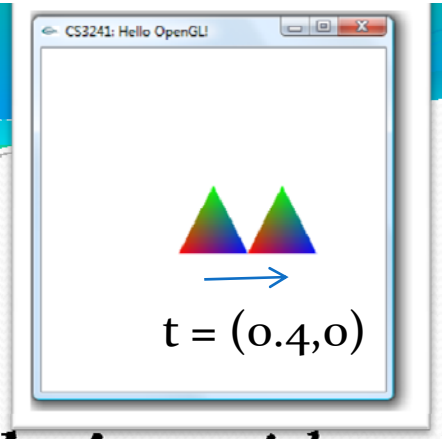
Naïve (Bad) Way

```
void drawASmallTri()
{
    glBegin(GL_POLYGON);
       glColor3f(1.0,0.0,0.0);
       glVertex2f(-0.2, -0.2);
       glColor3f(0.0,1.0,0.0);
       glVertex2f(0, 0.2);
       glColor3f(0.0,0.0,1.0);
       glVertex2f(0.2, -0.2);
    glEnd();
}
void mydisplay(){
    glClearColor(1.0,1.0,1.0,1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    drawASmallTri();
    drawAnotherSmallTri();
    glFlush();
}
```




```
void drawAnotherSmallTri()
{
    glBegin(GL_POLYGON);
       glColor3f(1.0,0.0,0.0);
       glVertex2f(0.2, -0.2);
       glColor3f(0.0,1.0,0.0);
       glVertex2f(0.4, 0.2);
       glColor3f(0.0,0.0,1.0);
       glVertex2f(0.6, -0.2);
    glEnd();
}
```

Translation



- We called this kind of movement a **translation** with a translation vector $t = (t_x, t_y)$
  - In the previous case $t = (0.4,0)$
- Every vertex of the triangle adds the vector t
- With homogeneous coordinates, every vertex (x,y) will be moved to a new position (x',y') by

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

New position

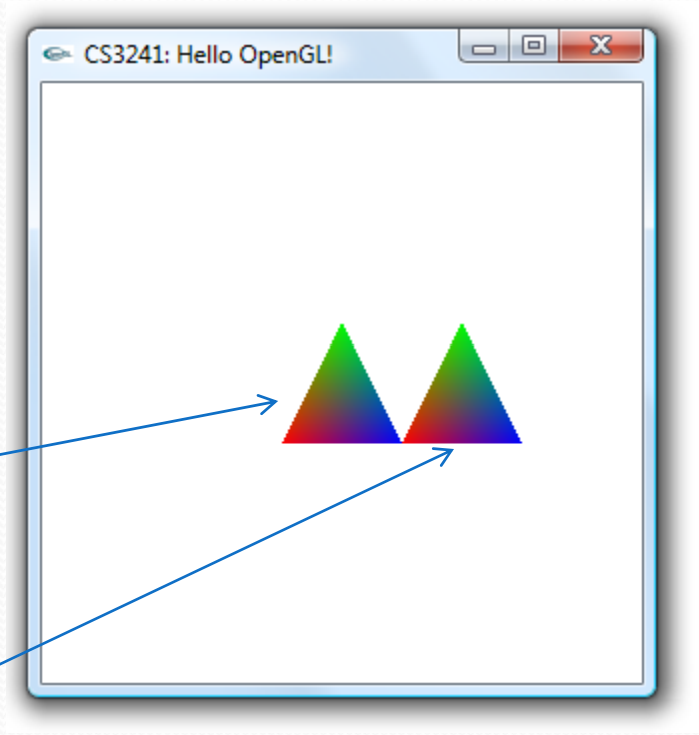
Old position

glTranslate(x,y,z)

```
void mydisplay(){
  glClearColor(1.0,1.0,1.0,1.0);
  glClear(GL_COLOR_BUFFER_BIT);

  glLoadIdentity();
  drawASmallTri();

  glTranslatef(0.4,0,0);
  drawASmallTri();

  glFlush();
}
```

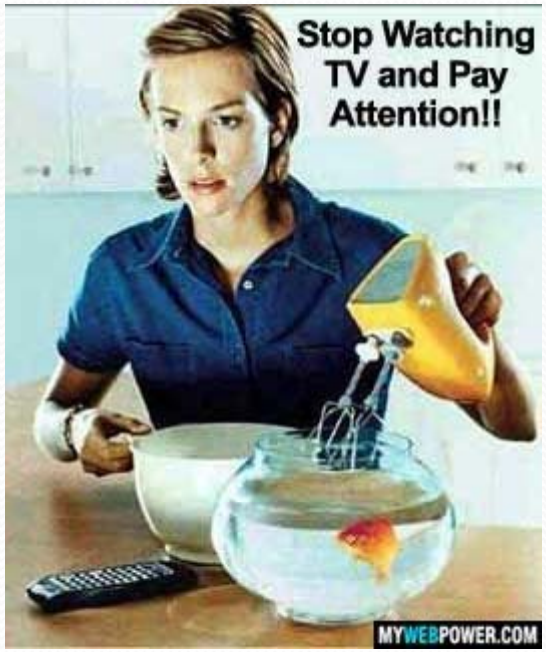
Original Triangle

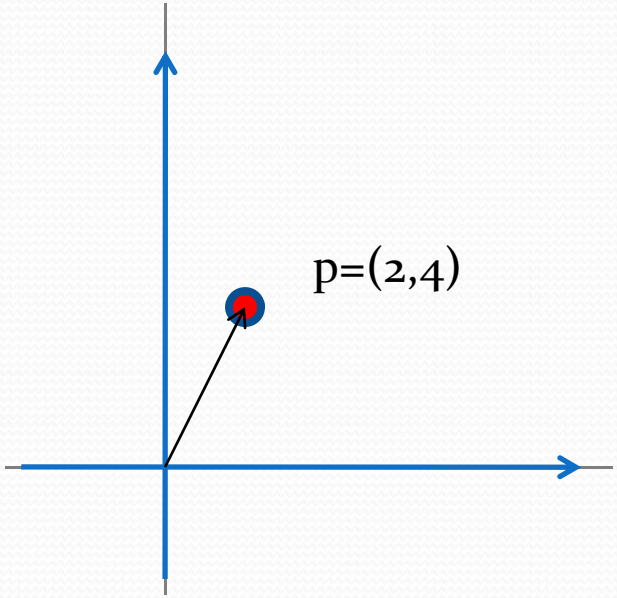
New Triangle



Reference Frame

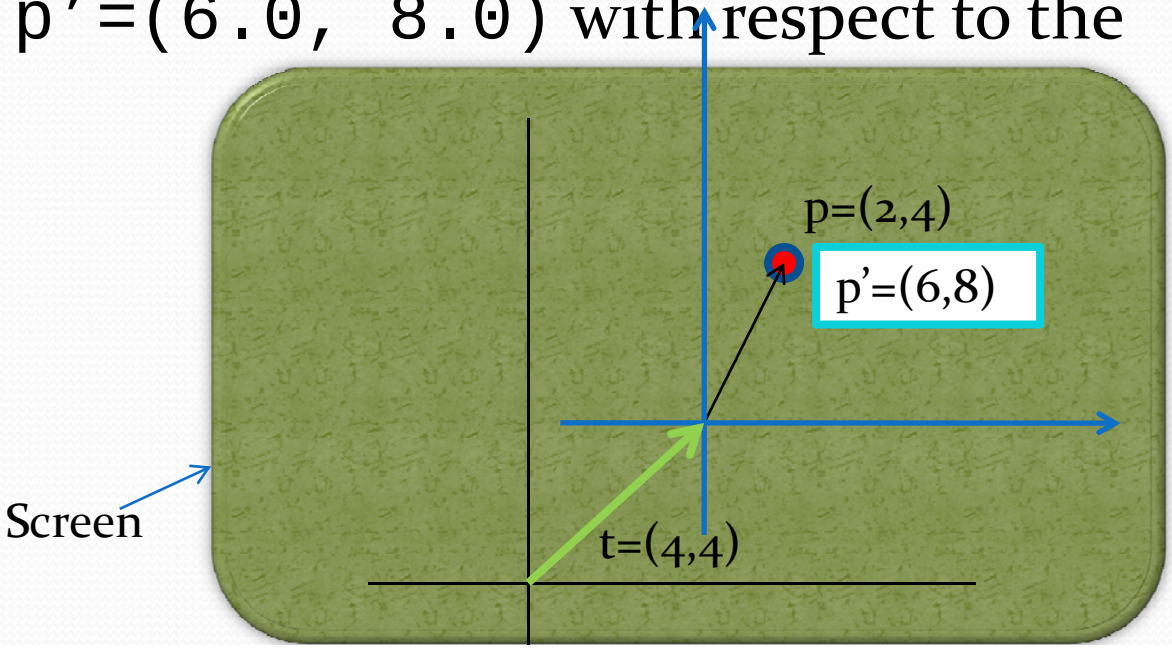
- For every coordinates, there is a **reference frame**
- E.g. p=(2,4) meaning p is 2 units right of and 4 units above the origin reference frame



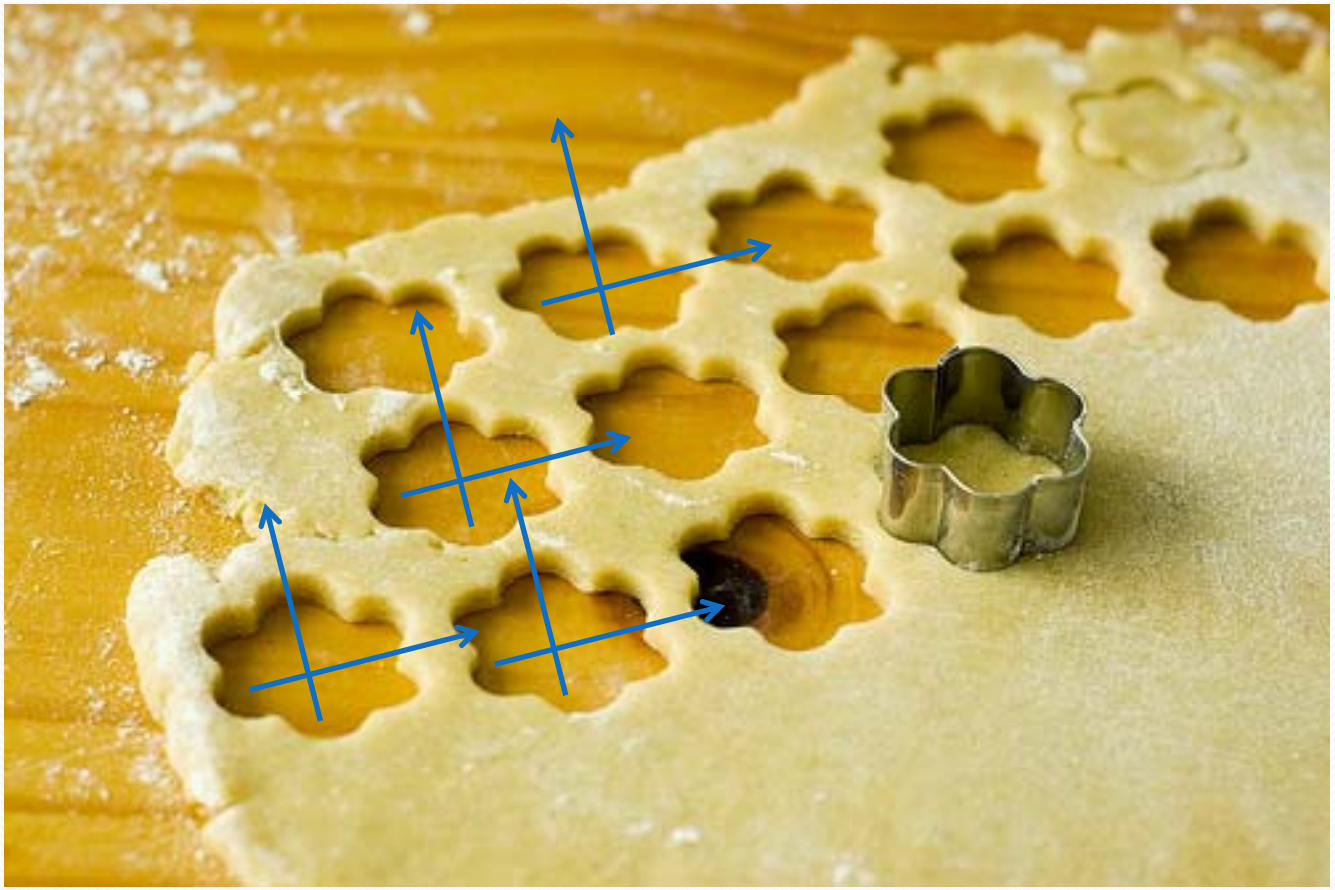


Reference Frame

- By applying a translation, it means moving the reference frame to a new position
- For example `glTranslatef(4.0,4.0,0)`
- Then when we draw `p` at `(2,4)` with respect to the **NEW** reference frame
- It finally becomes `p'=(6.0, 8.0)` with respect to the original frame

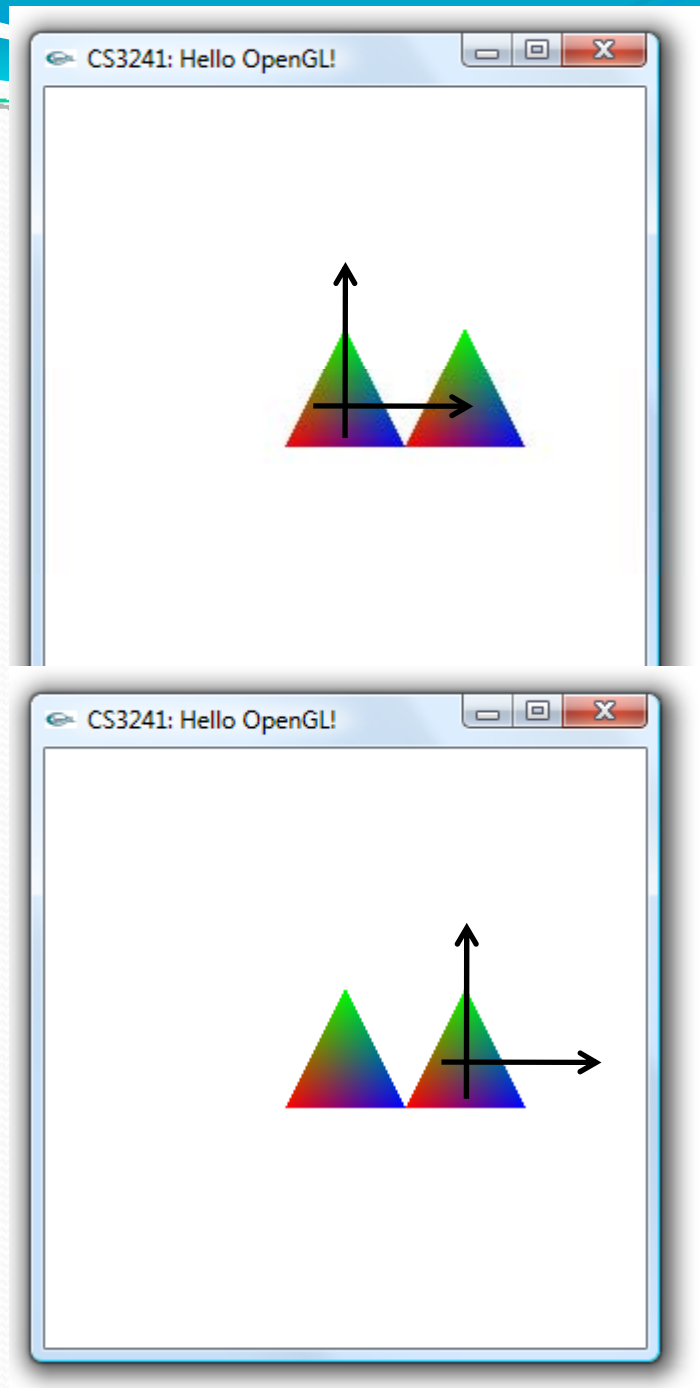


Reference Frame



glTranslate(x,y,z)

```
void mydisplay(){
  glClearColor(1.0,1.0,1.0,1.0);
  glClear(GL_COLOR_BUFFER_BIT);

  glLoadIdentity();
  drawASmallTri();

  glTranslatef(0.4,0,0);
  drawASmallTri();

  glFlush();
}
```



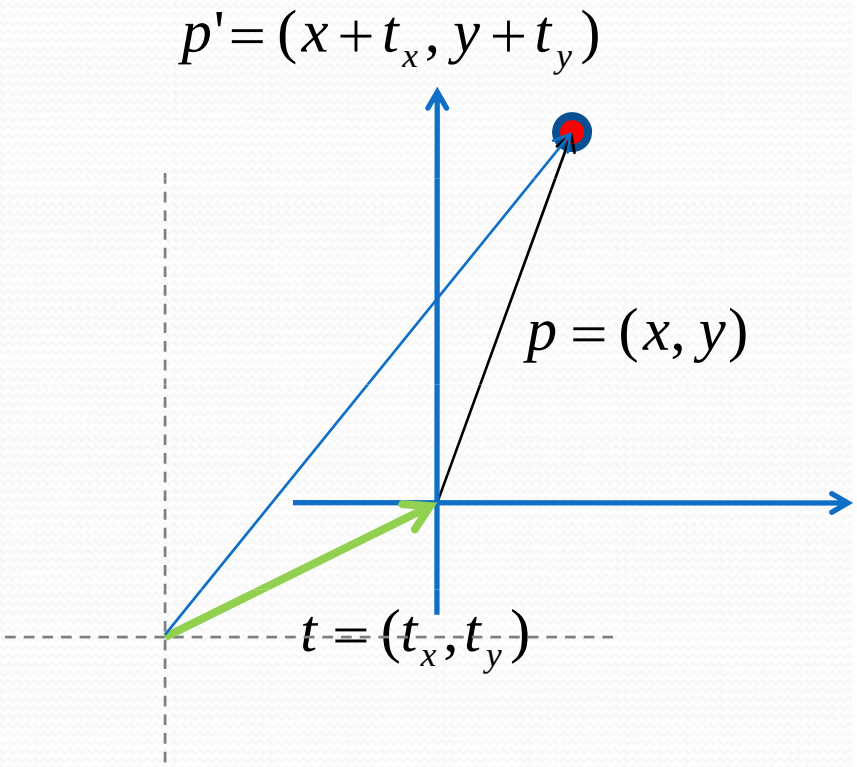
Translation

- Moving the **reference frame** by a vector

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

- By a **translation matrix**

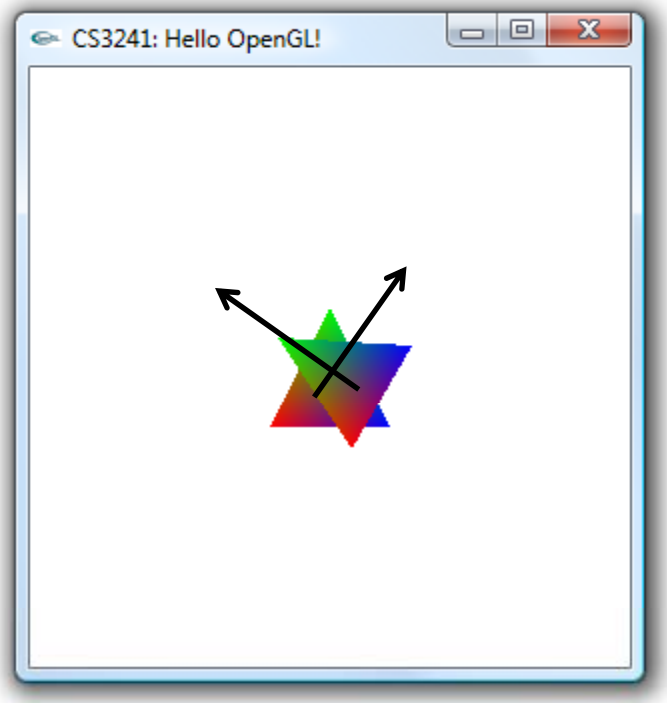
$$T(t_x, t_y) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$



glRotatef(angle,x,y,z)

```
void mydisplay(){
  glClearColor(1.0,1.0,1.0,1.0);
  glClear(GL_COLOR_BUFFER_BIT);

  glLoadIdentity();
  drawASmallTri();

  glRotatef(60,0,0,1);
  drawASmallTri();

  glFlush();
}
```

Actually a 3D rotation around the axis of the vector (0,0,1), namely the z-axis



Rotation

- Rotation all vertices around the origin
- Anti-clockwise

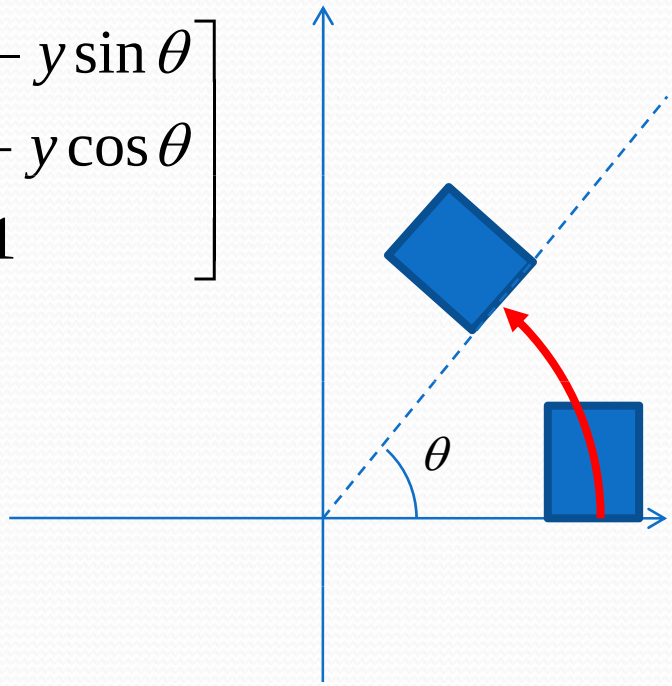
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \\ 1 \end{bmatrix}$$

Old points

New points

- By a **rotational matrix**

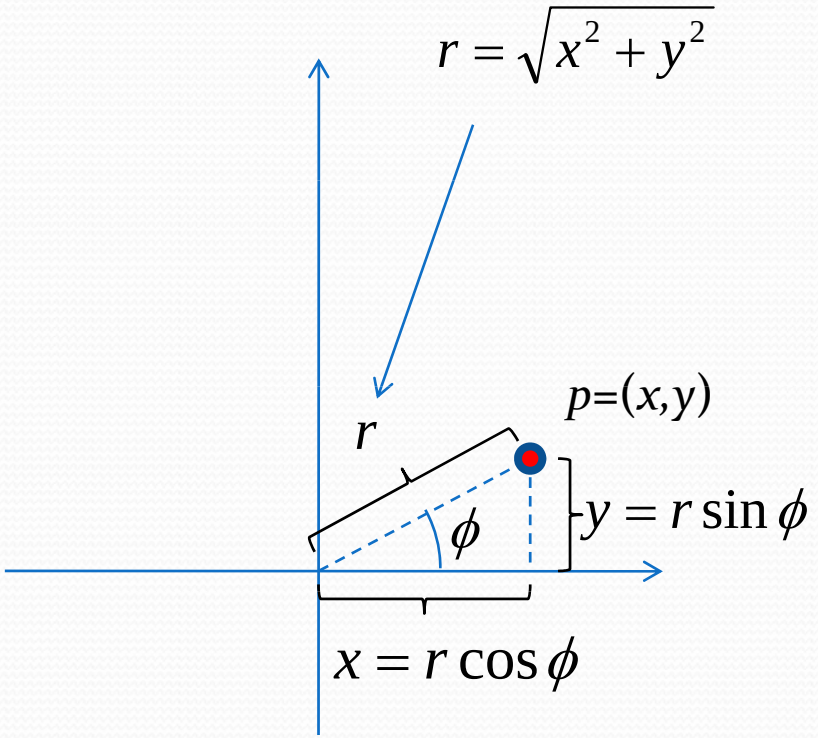
$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Rotation (Derivation)

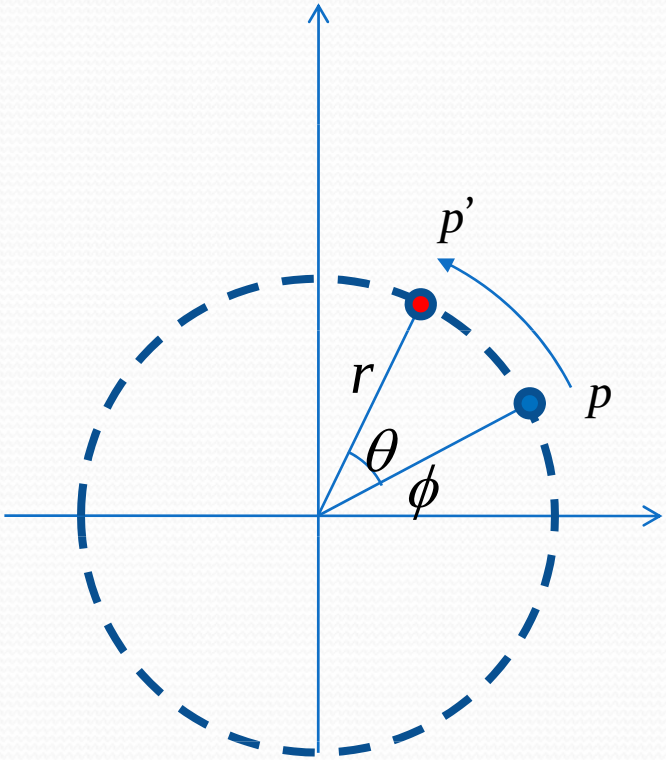
- Before rotation
- For any point $p = (x,y)$, we can represent p in the ***polar coordinates***

$$p = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} r\cos\phi \\ r\sin\phi \\ 1 \end{bmatrix}$$



Rotation (Derivation)

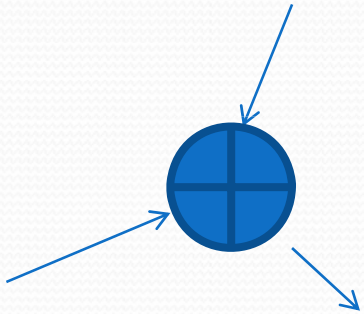
- Rotate by an angle θ
- The new point $p' = \begin{bmatrix} r\cos(\theta+\phi) \\ r\sin(\theta+\phi) \\ 1 \end{bmatrix}$



Rotation (Derivation)

$$p' = \begin{bmatrix} r\cos(\theta+\phi) \\ r\sin(\theta+\phi) \\ 1 \end{bmatrix} = \begin{bmatrix} r\cos\theta\cos\phi - r\sin\theta\sin\phi \\ r\sin\theta\cos\phi + r\cos\theta\sin\phi \\ 1 \end{bmatrix}$$

$$p = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} r\cos\phi \\ r\sin\phi \\ 1 \end{bmatrix}$$

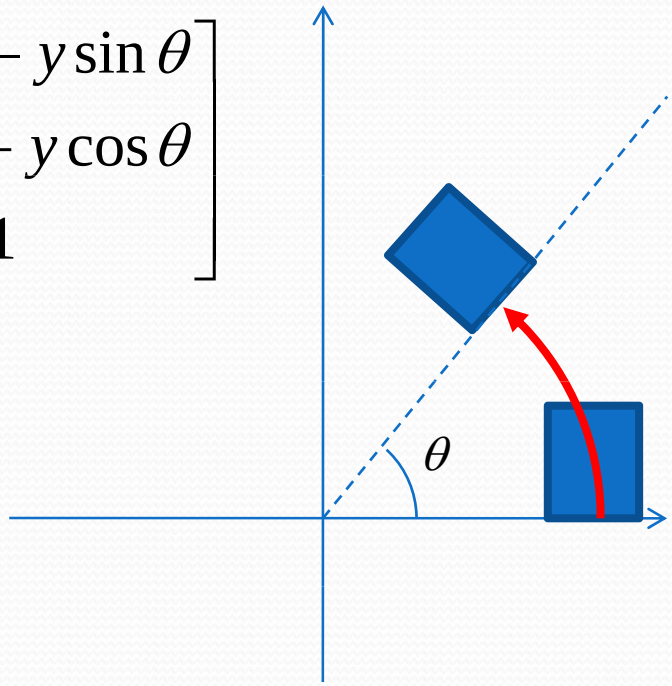


$$p' = \begin{bmatrix} r\cos(\theta+\phi) \\ r\sin(\theta+\phi) \\ 1 \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \\ 1 \end{bmatrix}$$

Rotation

- Rotation all vertices around the origin
- Anti-clockwise

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \\ 1 \end{bmatrix}$$

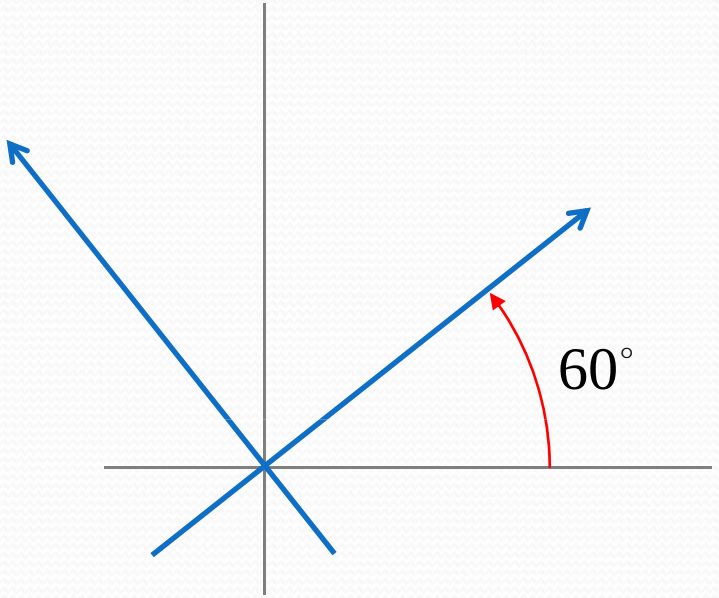


- By a **rotational matrix**

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

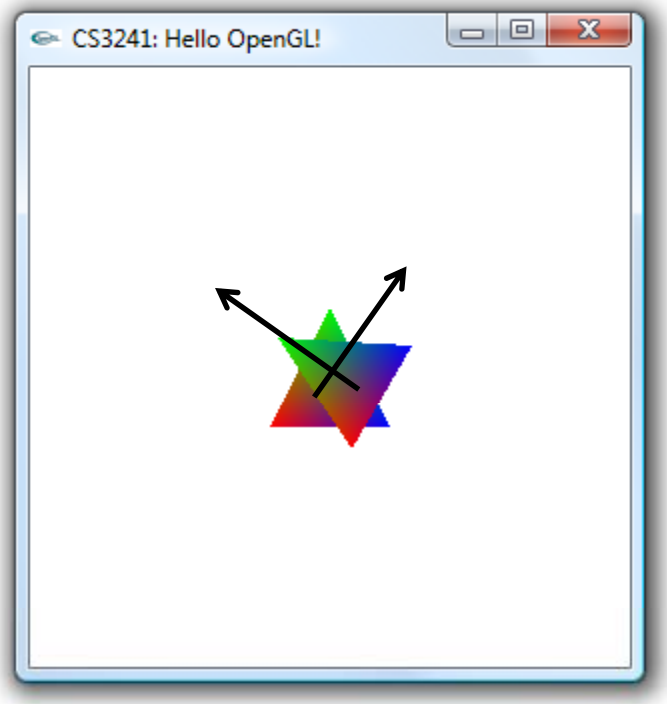
Reference Frame

- By applying a rotation, it means rotating the reference frame to a new orientation
- For example **`glRotatef(60.0,0,0,1)`**
  - Rotating 60 degree around the vector (0,0,1)



glRotatef(angle,x,y,z)

```
void mydisplay(){
  glClearColor(1.0,1.0,1.0,1.0);
  glClear(GL_COLOR_BUFFER_BIT);

  glLoadIdentity();
  drawASmallTri();

  glRotatef(60,0,0,1);
  drawASmallTri();

  glFlush();
}
```

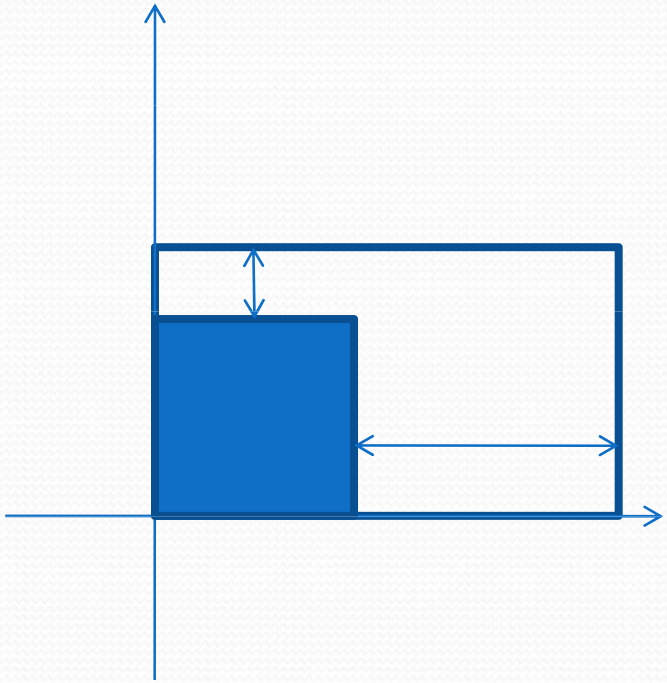


Scaling

- Shrink or grow
- Simply

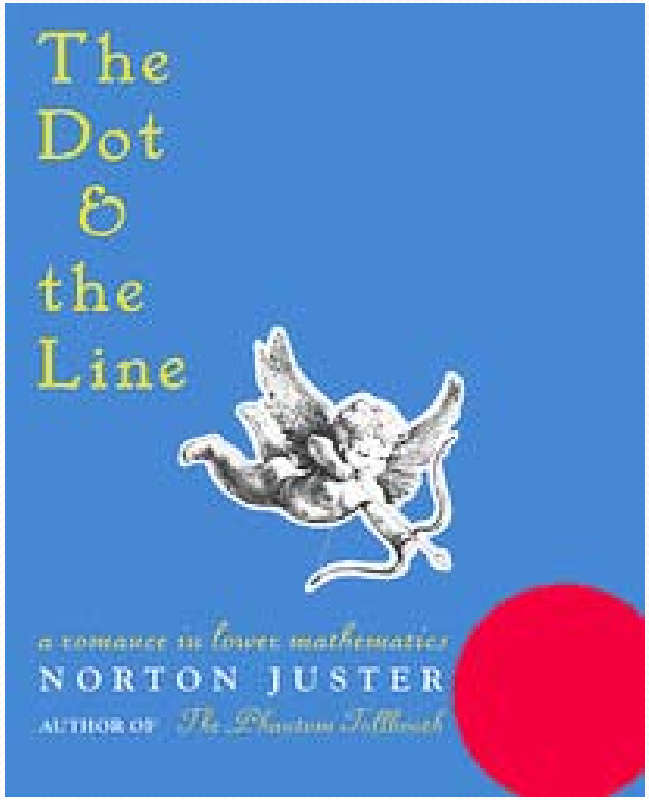
$$p' = \begin{bmatrix} ax \\ by \\ 1 \end{bmatrix} = S(a,b) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- a and b could be -ve
- By a scaling matrix $S(a,b) = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix}$

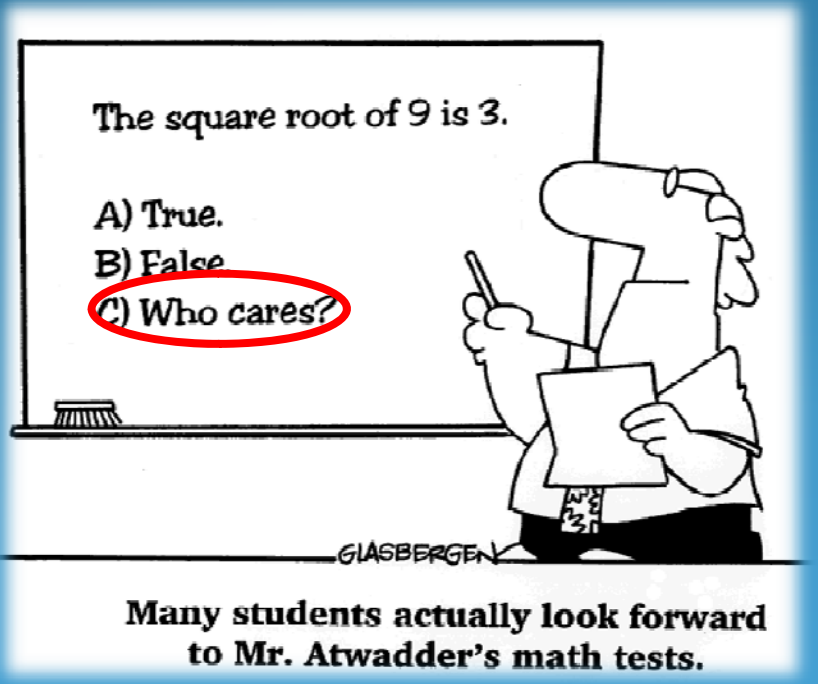


- OpenGL function:
  - **`glScalef(x,y,z);`**

Let’s take a break

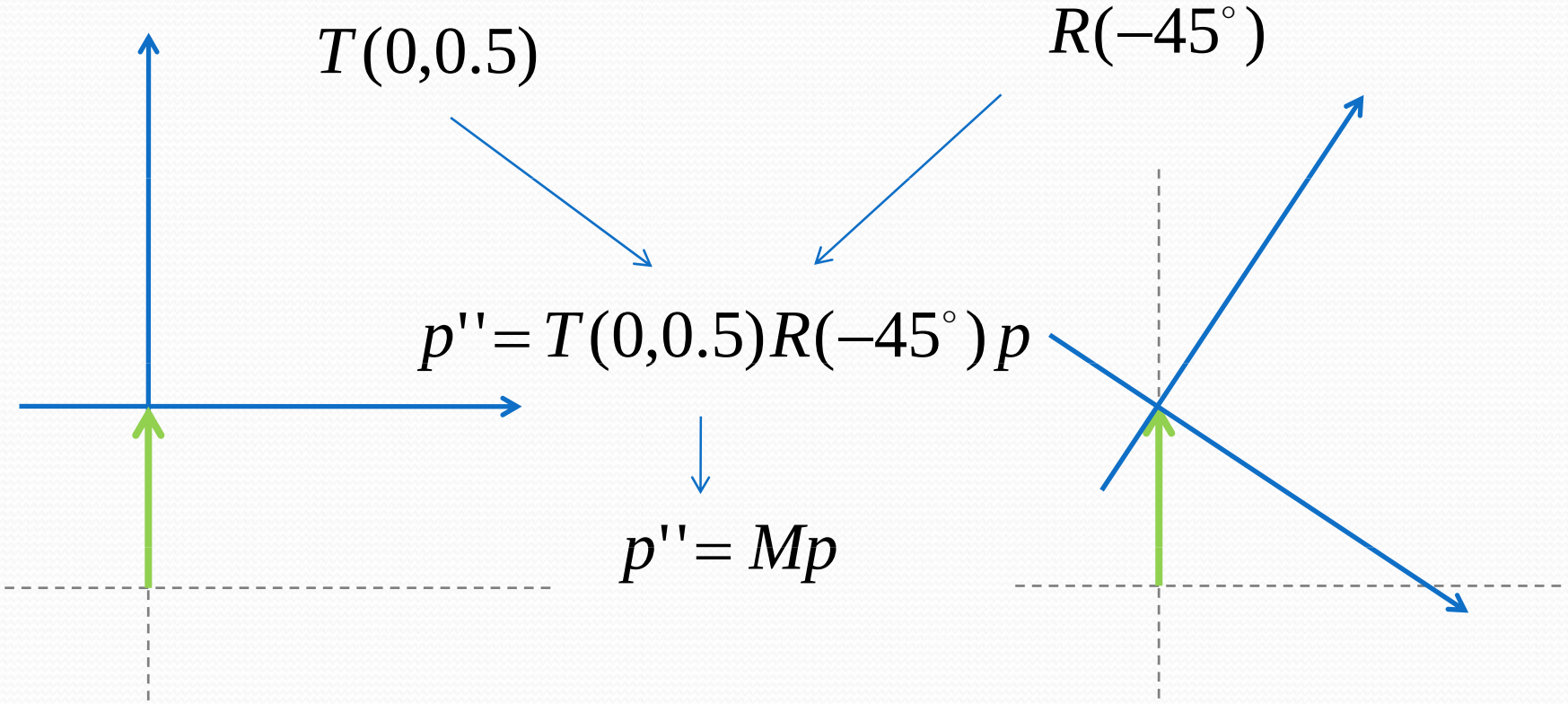


Why Matrix?


The square root of 9 is 3.
A) True.
B) False
C) Who cares?
GLASBERGEN
Many students actually look forward to Mr. Atwadder's math tests.

Why Matrices??

- Because we can combine **many** transformations into one matrix



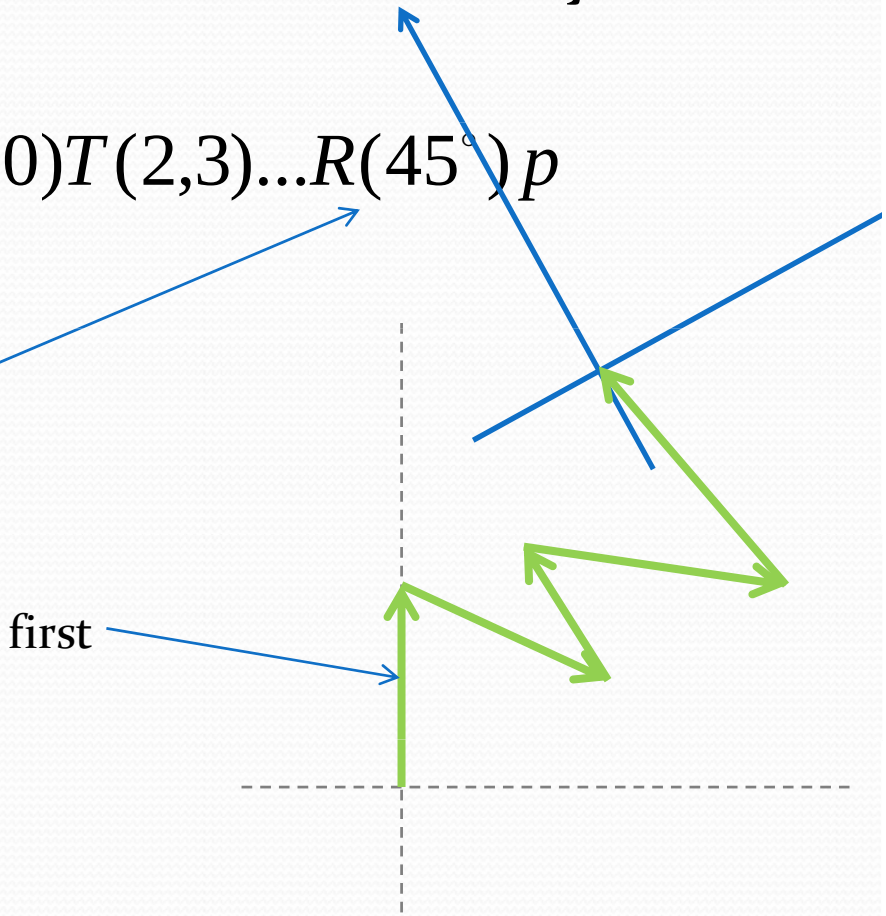
Composite Transformation

- Note that you can do as many transformation as you can

$$p'' = T(0,1)R(10^\circ)S(0.5,20)T(2,3)\ldots R(45^\circ)p$$

first

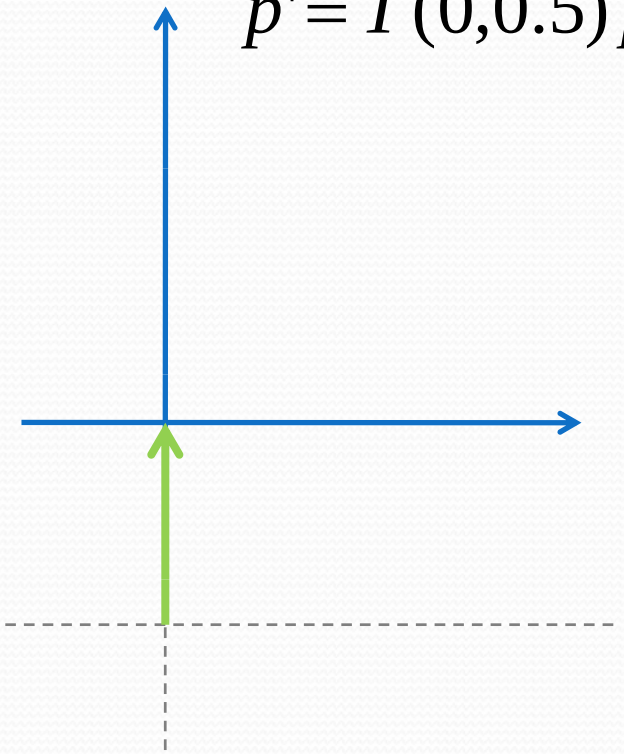
- And also note the “order”



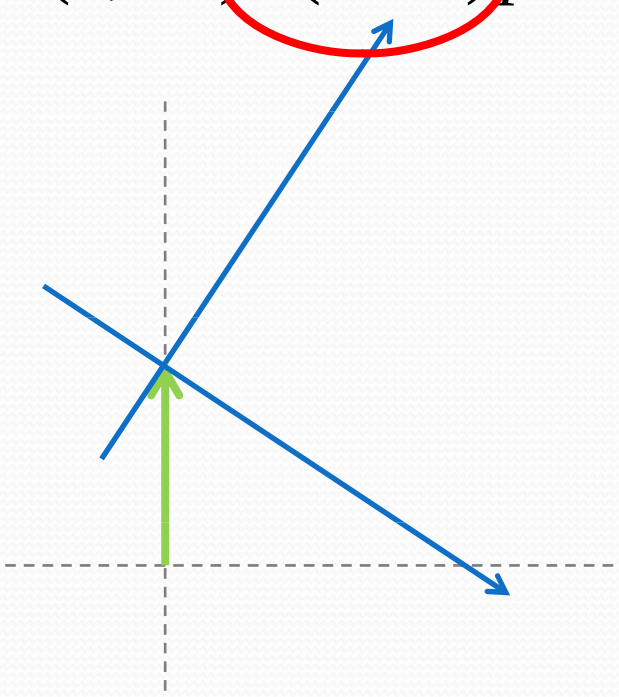
Composite Transformation

- Whenever you add a new transformation, you add it to the RIGHT of the matrices

$$p' = T(0,0.5)\,p$$



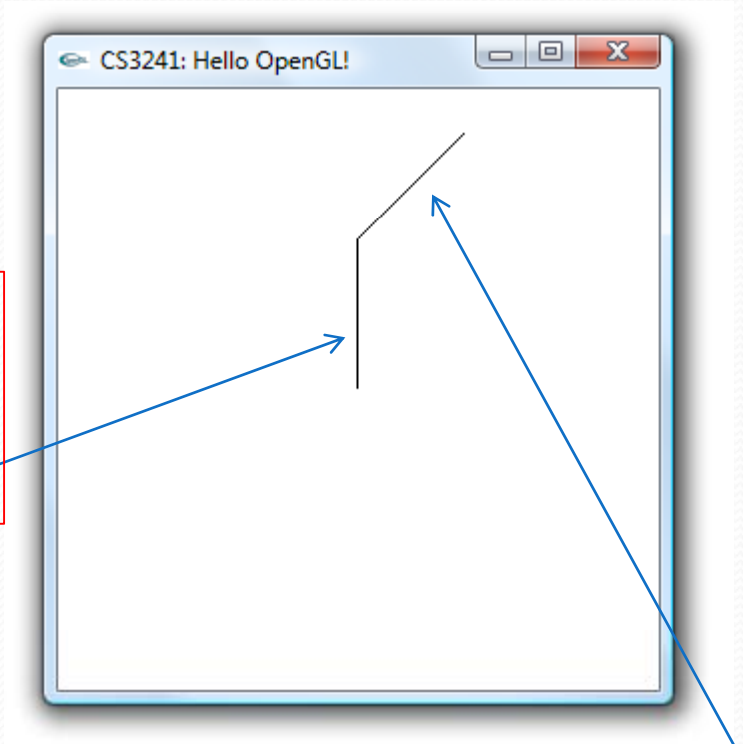
$$p'' = T(0,0.5)R(-45^\circ)\,p$$



In OpenGL

```
void drawLine()
{
   glBegin(GL_LINES);
         glVertex2f(0,0);
         glVertex2f(0,0.5);
   glEnd();
}

void mydisplay(){
  …

  glLoadIdentity();

  drawLine();
  glTranslatef(0,0.5,0);
  glRotatef(-45,0,0,1);
  drawLine();
  glFlush();
}
```

Clear the transformation matrix to identity



Matrix Stack

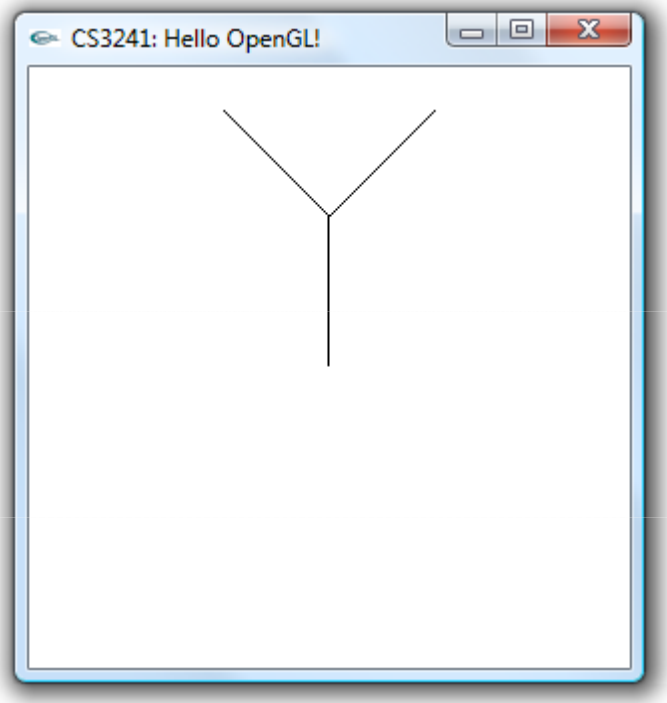
What if I want to draw this

- Draw the first vertical line
- Do two transformations and draw the right line:

$$T(0,0.5)R(-45)$$

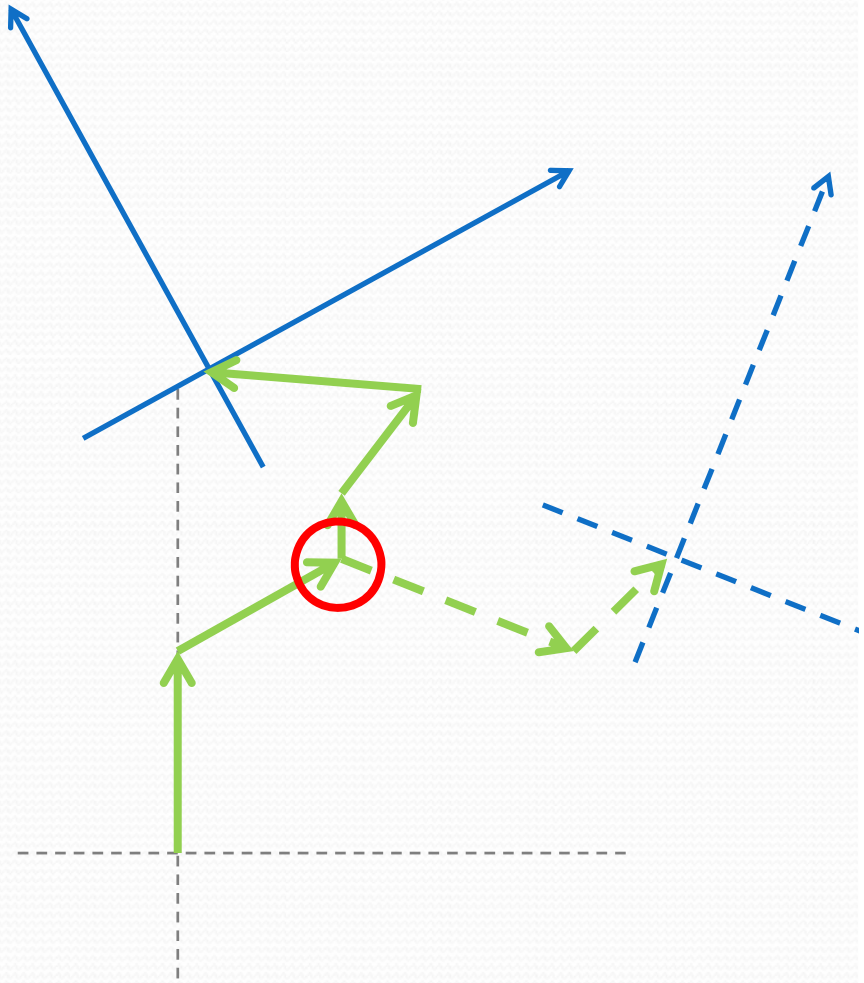
- Reset the transformation
- Do another two transformations to draw the left line

$$T(0,0.5)R(45)$$



Transformation Stack in OpenGL

- You can save the transformation in any moment into a stack
- From there you can "resume" or "save" at anytime and add on new transformation
- Save a matrix
  - **`glPushMatrix()`**
- Retrieve a matrix
  - **`glPopMatrix()`**

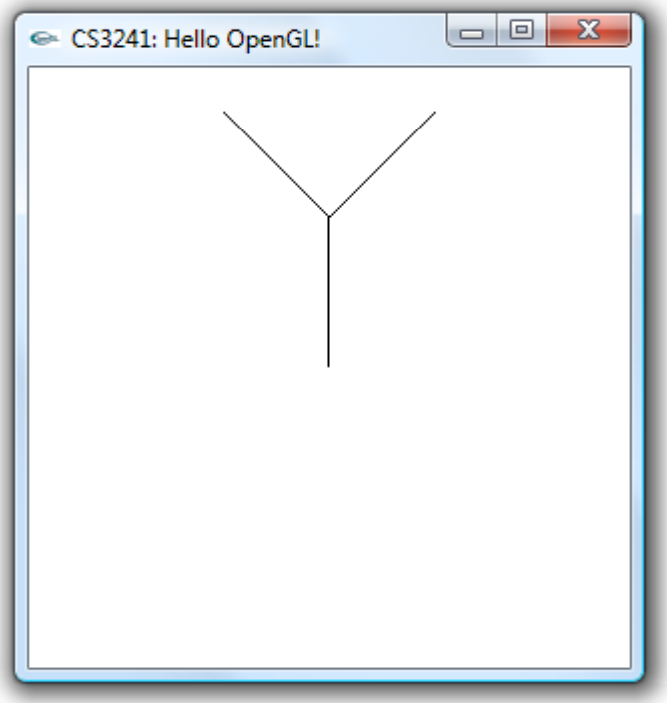


In OpenGL

```
void drawFork()
{
   drawLine();
   glPushMatrix();
          glTranslatef(0,0.5,0);
          glPushMatrix();
                   glRotatef(-45,0,0,1);
                   drawLine();
          glPopMatrix();
          glPushMatrix();
                   glRotatef(45,0,0,1);
                   drawLine();
          glPopMatrix();
   glPopMatrix();
}

void mydisplay(){
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(0,0,0);
   glMatrixMode(GL_MODELVIEW);
   drawFork();
   glFlush();
}
```

Resume to T(0,0.5)

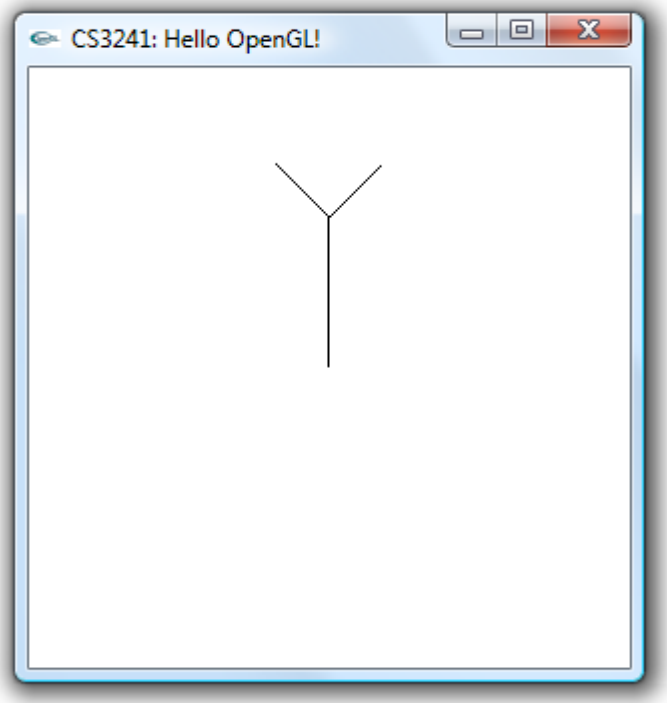
Resume to nothing



A Nicer "Y"

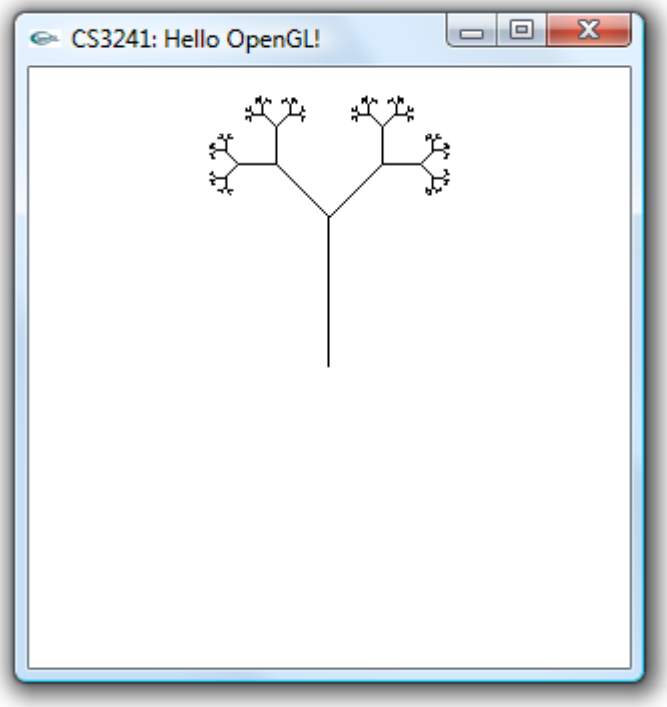
You can mess up the transformation within a pair of Push/Pop

```
void drawFork()
{
   drawLine();
   glPushMatrix();
         glTranslatef(0,0.5,0);
         glPushMatrix();
                  glRotatef(-45,0,0,1);
                  glScalef(0.5,0.5,0.5);
                  drawLine();
         glPopMatrix();
         glPushMatrix();
                  glRotatef(45,0,0,1);
                  glScalef(0.5,0.5,0.5);
                  drawLine();
         glPopMatrix();
   glPopMatrix();
}
```



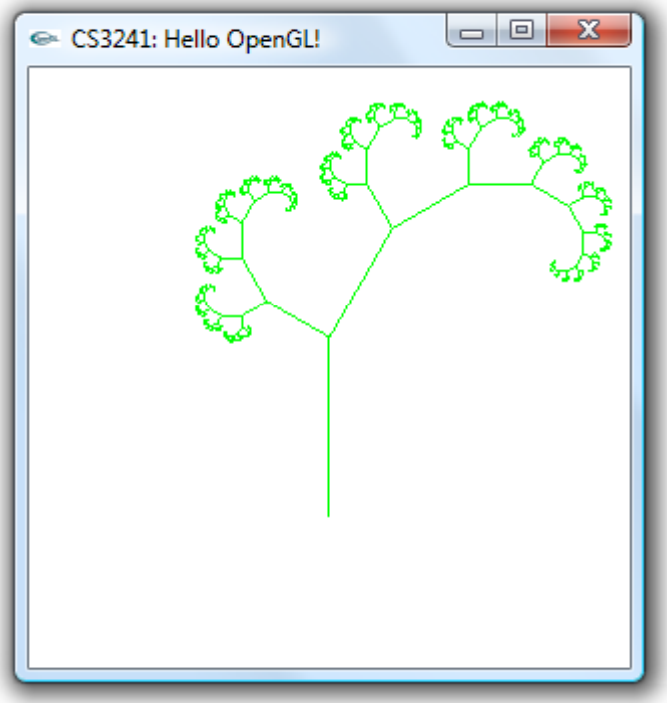
A Fractal Tree

```
void drawFork(int n)
{
   drawLine();
   if (n==0) return;
   glPushMatrix();
         glTranslatef(0,0.5,0);
         glPushMatrix();
                  glRotatef(-45,0,0,1);
                  glScalef(0.5,0.5,0.5);
                  drawFork(n-1);
         glPopMatrix();
         glPushMatrix();
                  glRotatef(45,0,0,1);
                  glScalef(0.5,0.5,0.5);
                  drawFork(n-1);
         glPopMatrix();
   glPopMatrix();
}
```

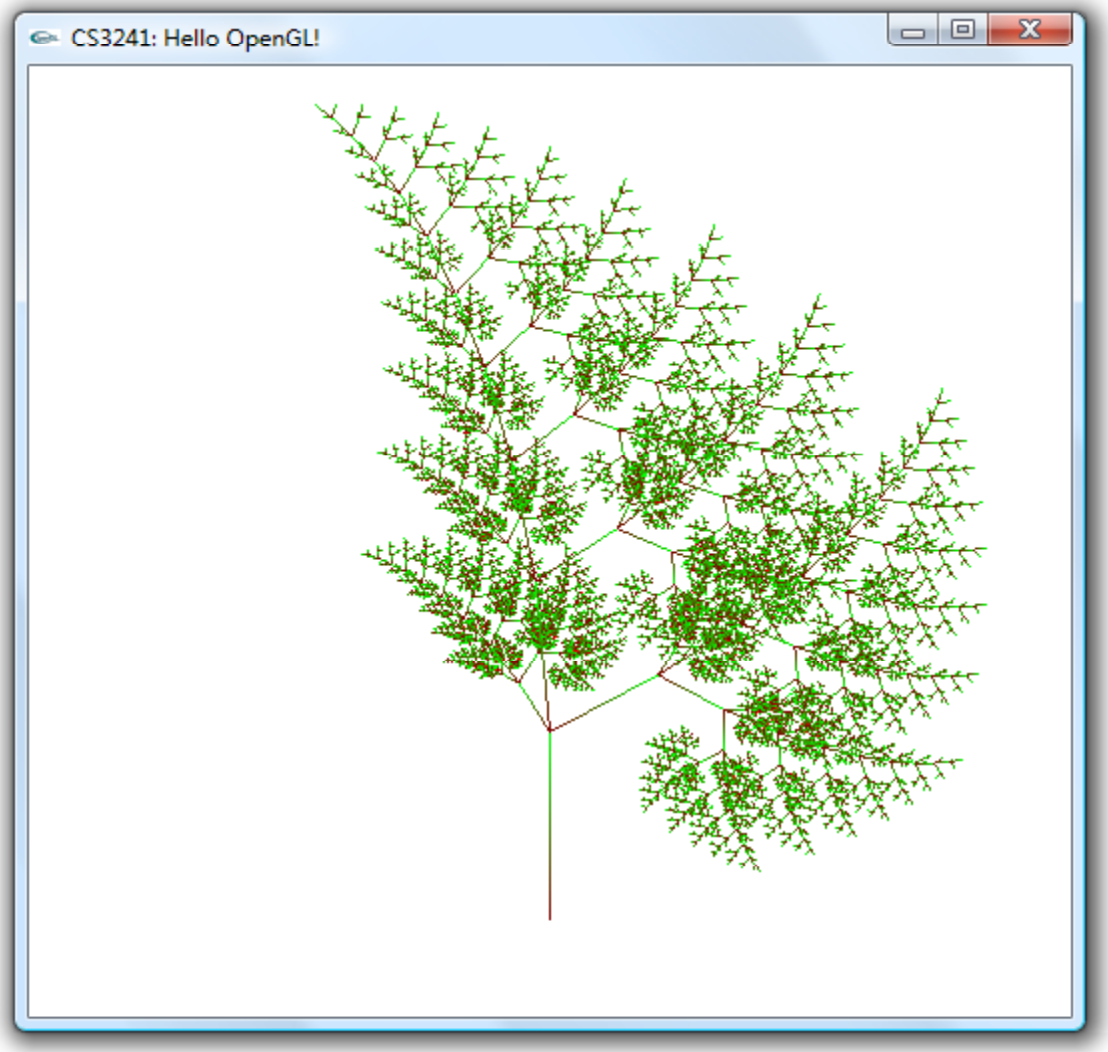


A Nicer Fractal Tree

```
void drawFork(int n)
{
   drawLine();
   if (n==0) return;
   glPushMatrix();
         glTranslatef(0,0.2,0);
         glPushMatrix();
                  glRotatef(-30,0,0,1);
                  glScalef(0.7,0.7,0.7);
                  drawFork(n-1);
         glPopMatrix();
         glPushMatrix();
                  glRotatef(60,0,0,1);
                  glScalef(0.4,0.4,0.4);
                  drawFork(n-1);
         glPopMatrix();
   glPopMatrix();
}
```



Finally, a colorful nicer fractal tree



Some Final Note

About Transformation

- Transformation is NOT effective within a pair of `glBegin()` and `glEnd()`
  - GL_INVALID_OPERATION is generated if `glTranslate` (and any other transformation) is executed between the execution of `glBegin` and the corresponding execution of `glEnd`.

Vector/Matrix Representations

- In **this** course, we will use a column matrix to represent a point

$$p = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad p'' = T(0,0.5)R(-45^\circ)\,p$$

- However, row matrices also work (in other textbooks)

$$q = \begin{bmatrix} x & y & 1 \end{bmatrix} = p^T \qquad q'' = qR(-45^\circ)^T T(0,0.5)^T = (p'')^T$$

Announcement

- Tutorials and Labs will start at NEXT week
- First assignment will be issued NEXT Tuesday
- For any confusion, please refer to IVLE, lesson plan
- Remember there is a Facebook Page
  - https://www.facebook.com/NUSCG