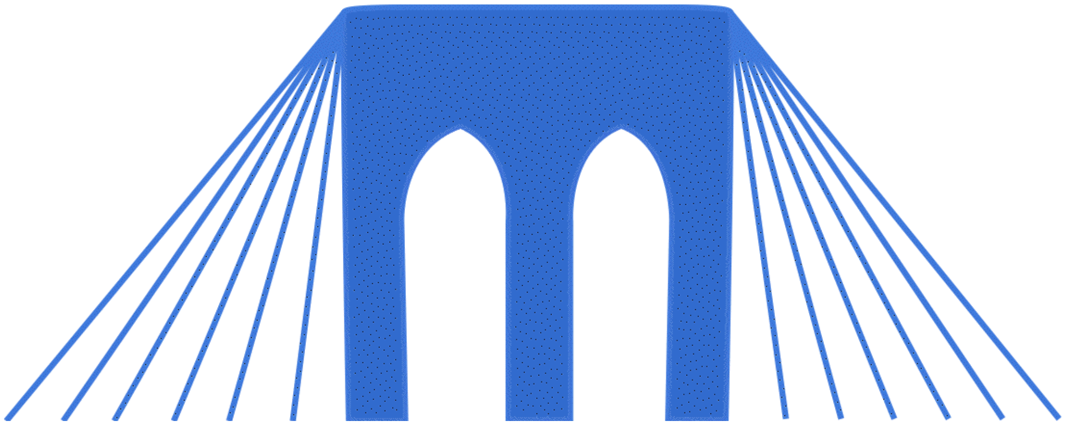
Bridges To Computing



General Information:

- This document was created for use in the "Bridges to Computing" project of Brooklyn College.

- For more information about the Bridges Program, please visit our website at: http://bridges.brooklyn.cuny.edu/

Disclaimers:

- **IMAGES**: All images in this presentation were created by our Bridges to Computing staff or were found online through open access media sites and are used under the Creative Commons Attribution-Share Alike 3.0 License. If you believe an image in this presentation is in fact copyrighted material, never intended for creative commons use, please contact us at http://bridges.brooklyn.cuny.edu/ so that we can remove it from this presentation.
- **LINKS**: This document may include links to sites and documents outside of the "Bridges to Computing" domain. The Bridges Program cannot be held responsible for the content of 3rd party sources and sites.

Graphics Programming

Lecture 3

- Simple Animation

- Simple 3D Drawing

Resources

- Processing web site:
  - http://www.processing.org/
- Linear motion (moving stuff around):
  - http://www.processing.org/learning/topics/linear.html
- Bitmap animation (swapping pictures):
  - http://www.processing.org/learning/topics/sequential.html
- Reference:
  - http://www.processing.org/reference/index.html
- MORE MORE MORE TUTORIALS
  - http://www.processing.org/learning/

Content

Animation

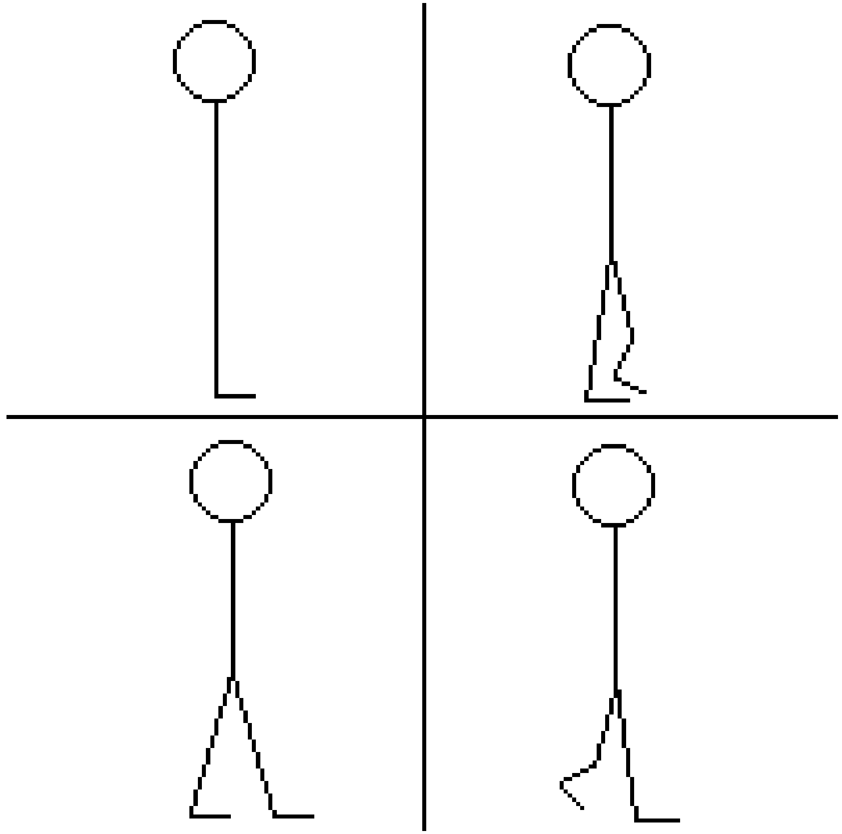
- Basic animation involves the following steps:
  1. Drawing initial frame - perhaps in setup().
  2. Waiting some amount of time (e.g., 1/60th of a second)
     - Processing does that automatically
  3. Erasing the screen.
     - Usually be reapplying the background (often the first thing we do in the draw( ) function).
  4. Drawing the next frame/image/picture.
  5. Repeating steps 2-4, until you are ready to stop animating.
- There are two basic ways to implement animation:
  - Drawing your own shapes, text, etc.
  - Displaying a GIF or other image file
- There are issues with the "frame rate" that we choose.

On Human Perception

- Human eyes and brains are unable to process properly images that are moving to fast.
- One famous example of this (which we can demonstrate) is the "flicker fusion" test.
  - A black box and a white box are alternately presented in the same exact place on the screen.
  - We change the frame rate so that those two boxes are alternated at faster and faster speeds.
  - Depending on the ambient light, at somewhere between 30 and 60 frames per second you will see a only a single grey box.
- You may need to slow down your own animations to 20-30 frames per second in order for some aspects of your animation to be visible.
- Most movies are and games are animated around the 24-30 frames per second rate.

Bitmap Animation (1)

- In Bitmap Animation we have a series of pictures, each of which is slightly different.
- We can achieve the illusion of animation, by presenting those different pictures rapidly in succession.
- In practical terms we will want to store our images in an array (a type of list).

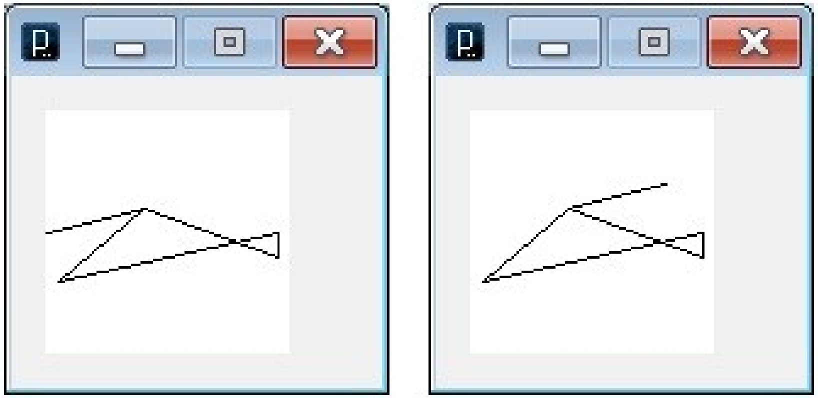


Bitmap Animation (2)

```
int numFrames = 4;      // The number of frames in the animation
int frame = 0;          // The picture to draw

PImage[ ] images = new PImage[numFrames];   // List of picture objects

void setup() {
   size( 200, 200 );
   frameRate( 30 );
   images[0] = loadImage("PT_anim0000.gif");
   images[1] = loadImage("PT_anim0001.gif");
   images[2] = loadImage("PT_anim0002.gif");
   images[3] = loadImage("PT_anim0003.gif");
}
void draw() {
   frame = ( frame + 1 ) % numFrames; // Cycle through frames
   image( images[frame], 50, 50 );
}
```

VECTOR ANIMATION (1)

- In Vector Animation we change the pictures that are drawn each frame by subtly modifying our mathematical code.
- In the example to the right, only one line of code is changed, in order to get the two different pictures.
- We can use a "gate" variable, to switch between drawing the different pictures each time.



Vector Animation (2)

```
int v_image=0;

void setup() {
   frameRate(20);
}

void draw() {
   background(#FFFFFF);    // clears the screen

    quad( 40,40, 95,60, 95,50, 5,70);    //body of the helicopter

   if(v_image == 0)  {
           line(40,40, 80,30);     // Line to the right
           v_image = 1;
    } else {
           line(40,40, 0,50);      // Line to the left
           v_image = 0;
    }
}
```

Animation and Movement

- Whether using a bitmap or a vector image, you will specify x and y coordinates for your image.
- If you add a variable to that coordinate you can change where an image is drawn.

```
int xPos = 0;     // x Position of our image.
void setup() {
   frameRate(20);
}
void draw() {
   background(#FFFFFF);
   rect( (0 + xPos) ,50 ,10 ,10 );
   xPos = xPos +1;
}
```

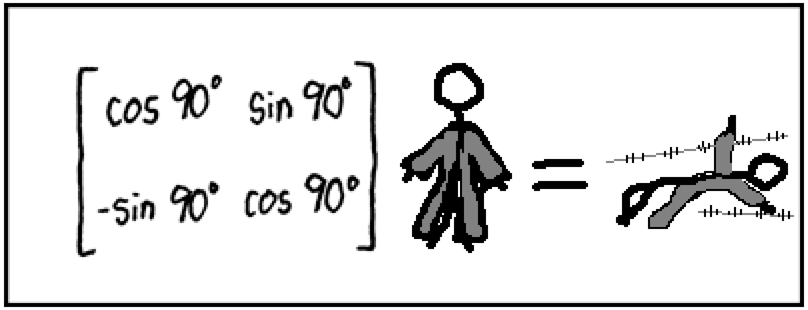
Animation and Movement (Vector)

```
int v_image=0;
int xPos = 0;     // x Position of our image.

void setup() {  frameRate(20);  }
void draw() {
    background(#FFFFFF);
    xPos = xPos -1;
    if(xPos < -95) {
            xPos = 95;
    }
    quad( 40+xPos,40,   95+xPos,60,   95+xPos,50,   5+xPos,70);
    if(v_image == 0) {
            line( 40+xPos,40,    80+xPos,30 );
            v_image = 1;
    } else {
            line( 40+xPos,40,    0+xPos,50 );
            v_image = 0;
    }
}
```

Animation and Movement (Bitmap)

```
int numFrames = 4;
int frame = 0;
int xPos = 0;
PImage[] images = new PImage[numFrames];

void setup() {
 ... // Same as before... load the four images
}

void draw() {
    background(#ffffff);
    frame = ( frame + 1 ) % numFrames;
    image( images[frame], 50+xPos, 50 );
    xPos = (xPos + 5);
    if( xPos > width) {
            xPos = -100;
    }
}
```

What is the Matrix?



- Unfortunately, no one can be told what the matrix is… you have to see if for yourself.
- Just kidding! A matrix (in graphics) is a kind of table (it has rows and columns).
- Think about a complex vector image; you should realize that it is made up of a LOT of points (X,Y,Z). Points that I might want to:
  - Translate (move in the X,Y or Z plane)
  - Rotate (again in the X,Y or Z plane)
  - Skew/Stretch/Resize (again in the X,Y or Z plane)
- That's a LOT of operations, over a LOT of points, and would be SLOW and TIME CONSUMING.

Matrices

- Graphic artists (and mathematicians) figured out that it was easier (and much FASTER) to load all the changes they wanted to perform on an image into a single matrix (again a kind of table).
- They could then and multiply that one matrix against all the points of the vector image they are working with.
- There is a special formula for multiplying matrices, which is beyond the scope of this class.

- Translation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Scaling

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Rotation

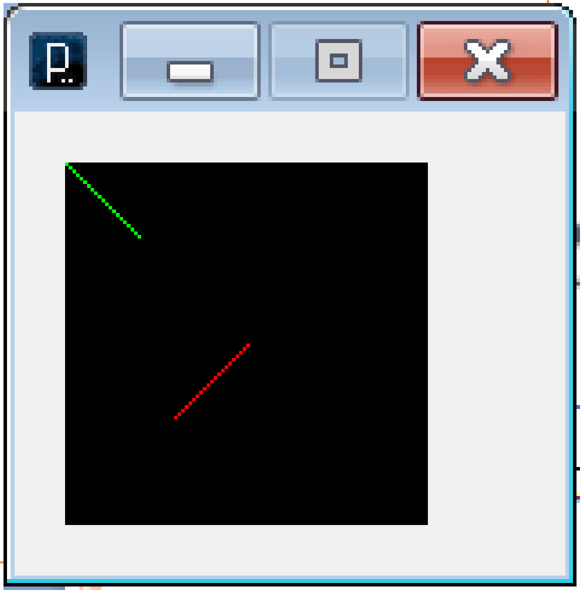
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Matrices in Processing

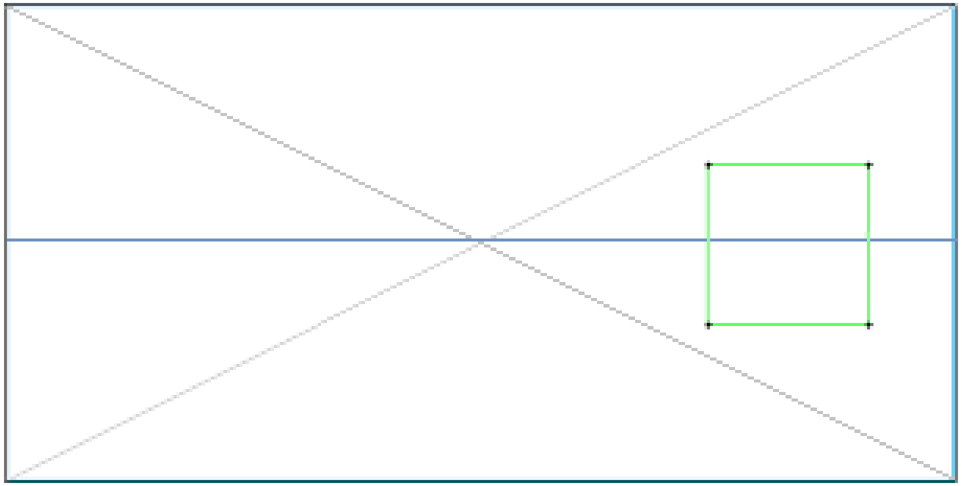
- You don't really need to worry about this... much.
- There are a bunch of neat operations (functions) that you can use to transform your images:
  - translate(x,y); // Will move your image
  - rotate(radians (a)); // Will rotate your image
- BUT YOU NEED TO BE CAREFUL!
  - When you perform the operations above you are changing the WORLD matrix. Which means that EVERYTHING you draw after a rotation() will be rotated.
  - To avoid this problem call pushmatrix() before you make any changes and popmatrix() after you have drawn what you wanted to change.

Processing Example

```
background(#000000);

pushMatrix();        // Save the state of the WORLD matrix

translate(50,50);       // Move the center of the WORLD to point 50,50
rotate( radians(90) );   // Rotate the WORLD by 90 degrees

stroke(#ff0000);        // Red line
line(0,0,20,20);

popMatrix();            // Reset the world

stroke(#00ff00);        // Green line
line(0,0,20,20);
```

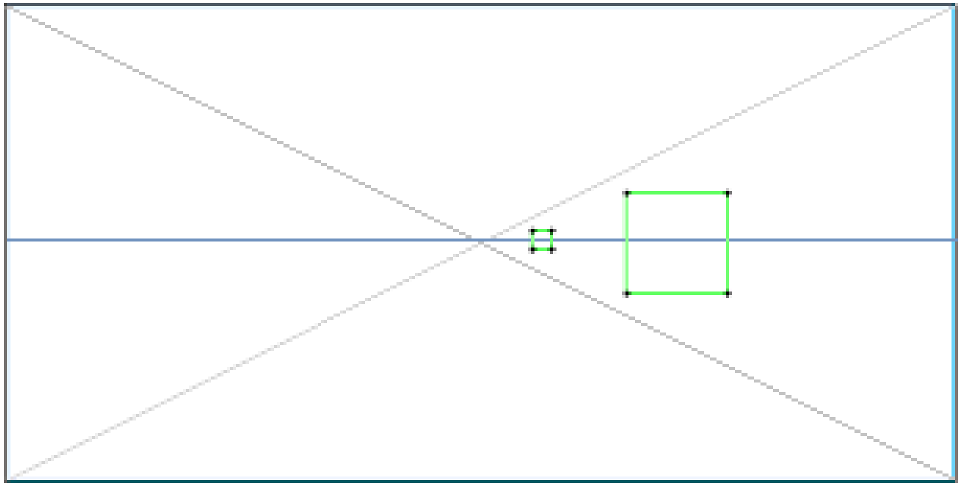


Simple 3D Animation (1)



- A vector image is made up of a series of connected points (x, y).
- In the real world a point has an x, y and **z** position.
- What happens to a point as it moves away from you?
- To put it another way, what will happen to the green box in the picture as it moves down the hall.

Simple 3D Animation (2)



- As the z value of all points increase (as the point move further into the distance) the x and y values will converge towards a single point (the focal point).
- The result of the convergence of the x and y points converging is that any images drawn using those points seem to get smaller.
- A 6' tall man remains six feet tall as he walks away, but takes up a smaller percentage of your field of view.

Simple 3D Animation (3)

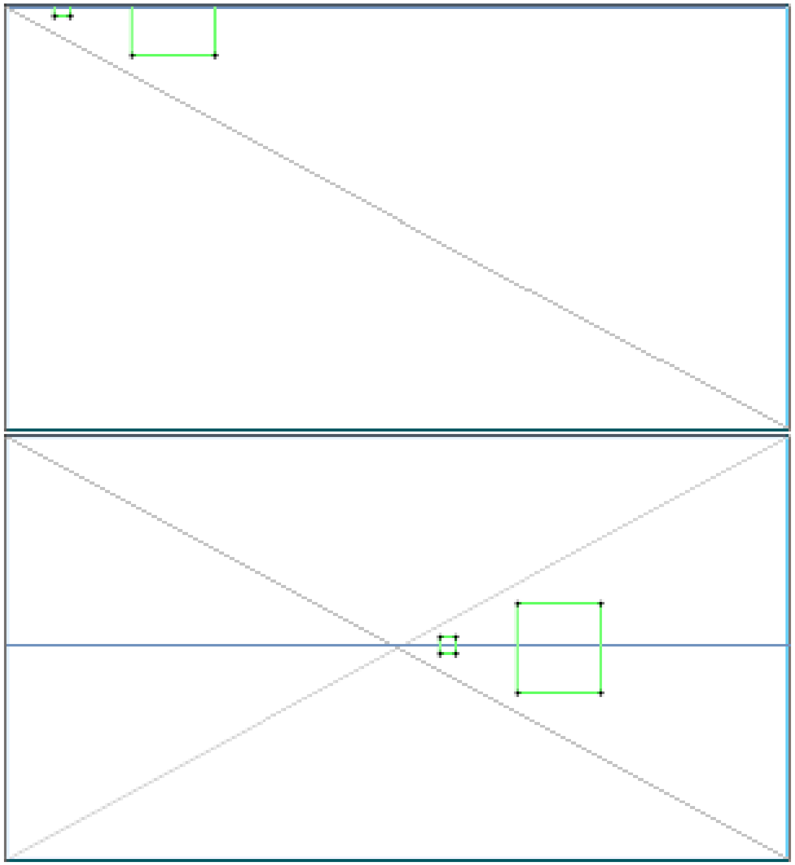
- To make an image appear to have 3 dimension or be able to move in 3 dimensions we must be able to construct it using a set of 3D points (x,y,z).
- We will also need:
  - Focal Point -> The place on the screen that all points will to converge too (usually specified as an offset).
  - Focal Length -> How far something can go in the z plane before it shrinks into nothing.
  - Px and Py -> To new variables that will represent the x and y positions of a point as they will actually appear on screen.
- We can then use the following formulas:

Px = (x * ((focalLength - z) / focalLength)) + offsetX;

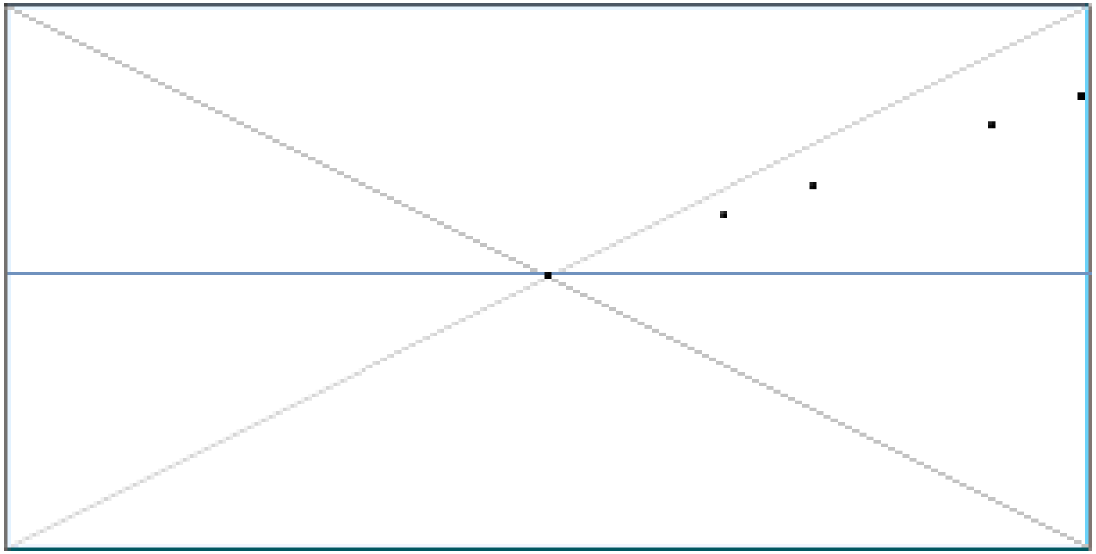
Py = (y * ((focalLength - z) / focalLength)) + offsetY;

Offset & Focal Point

- To keep the math simple we would like (0,0) to be our focal point.
- Unfortunately (0,0) is the top left of the screen.
- We can get around this problem by creating variable called offsetX and offsetY and adding them to all points.
- This has the effect of shifting the focal point.



Calculating Px and Py



Screen -> 300x150
offsetX -> 150
offsetY -> 75
focalLength ->300

x = 150
y = - 50
z = 0

Px = ?
Py = ?

- Formulas:

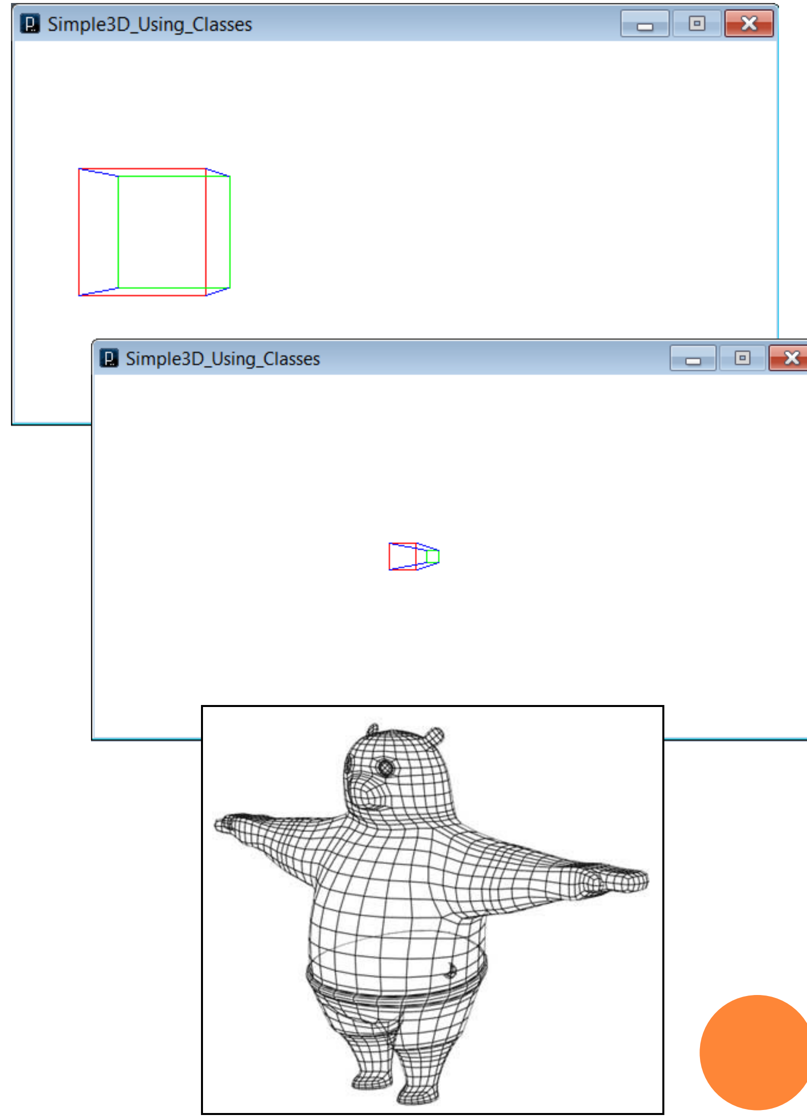
  Px = (x * ((focalLength - z) / focalLength)) + offsetX;

  Py = (y * ((focalLength - z) / focalLength)) + offsetY;

z	0	50	150	200	300
Px	300	375	225	200	150
Py	25	33	50	58	75

3D WireFrame Images

- A complex "wireframe" image is simple a vector image made up of lots of points, that are represented in the 3D
- Once you have the "wireframe" you can then color the areas between points and lines by applying "texels" which are usually bitmap images.
- Applying texels requires a little more math (but not much).



The End