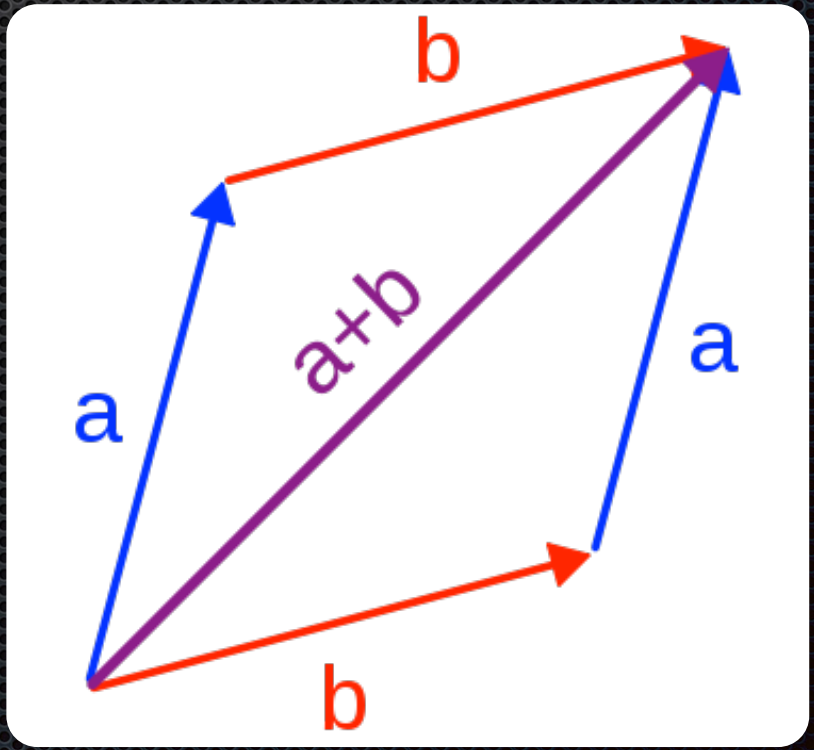
Mobile Application Programming

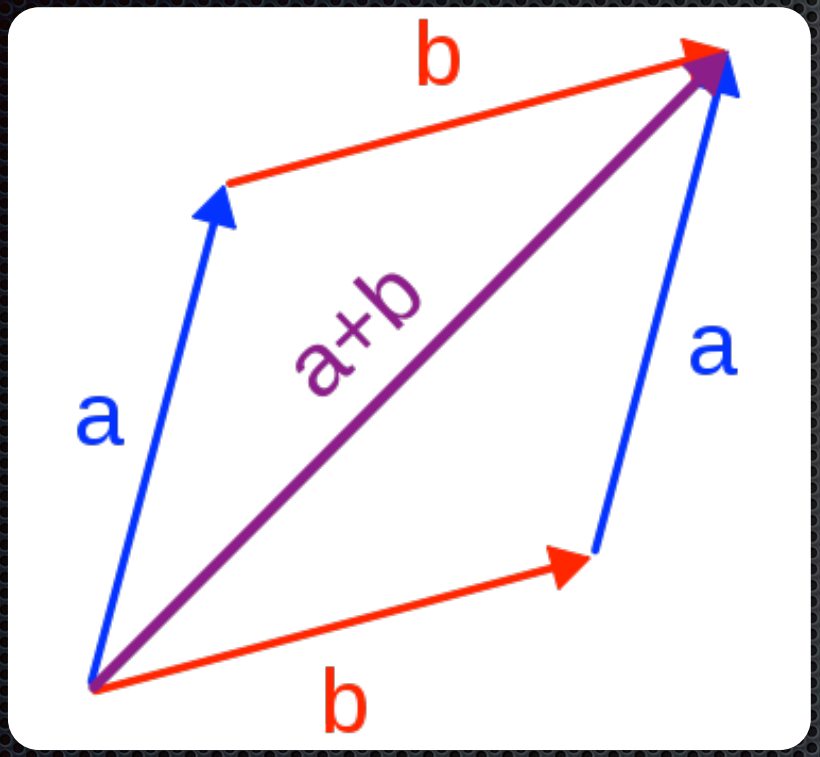
OpenGL ES 3D

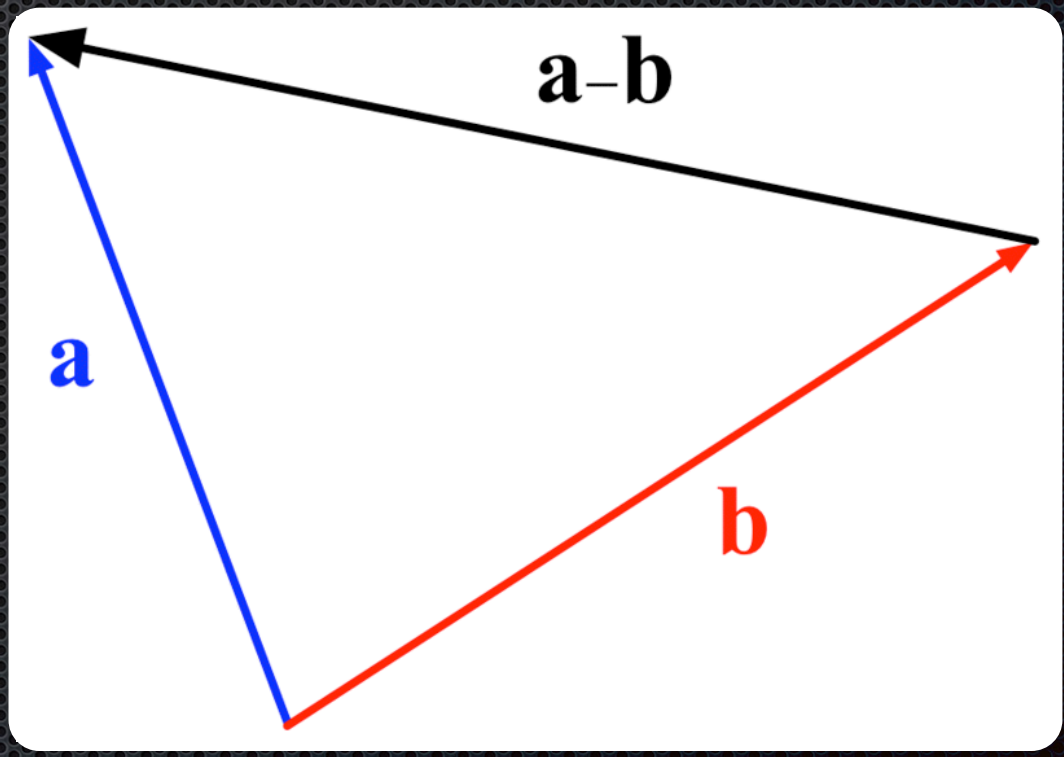
Vectors

- Vectors
  - Addition & Subtraction
  - Scalar Multiplication
  - Magnitude & Normalization
  - Dot & Cross Product

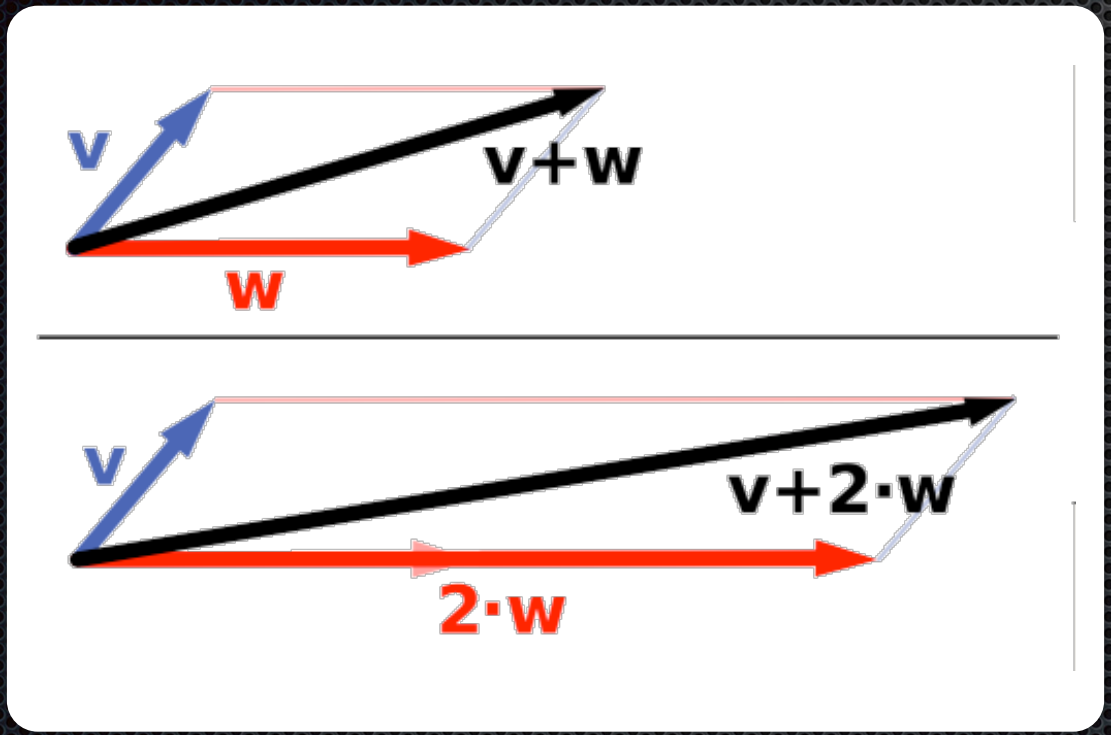


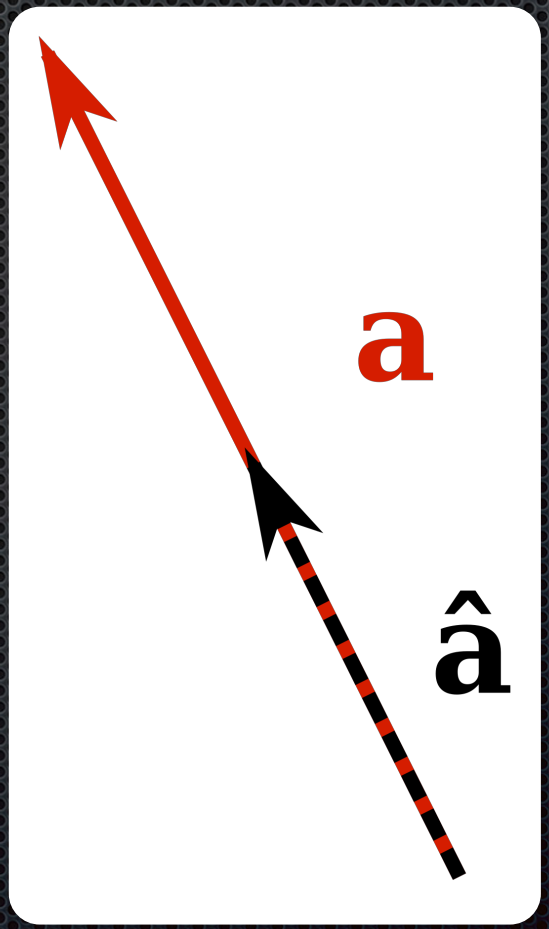
Vector Addition & Subtraction





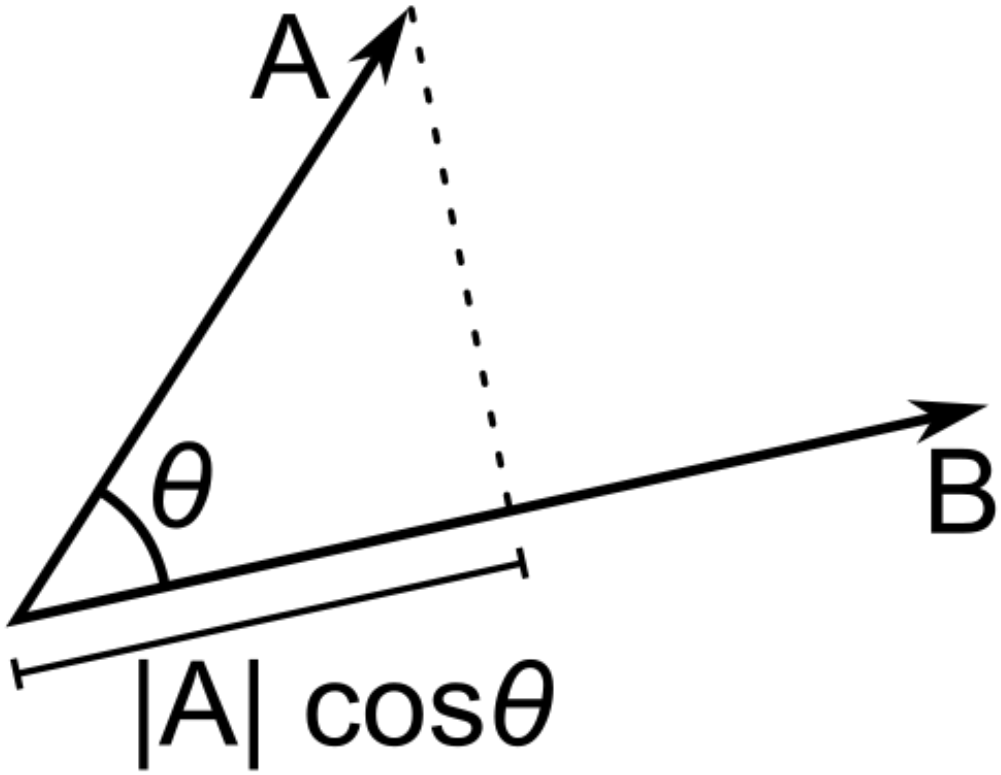
Vector-Scalar Multiplication



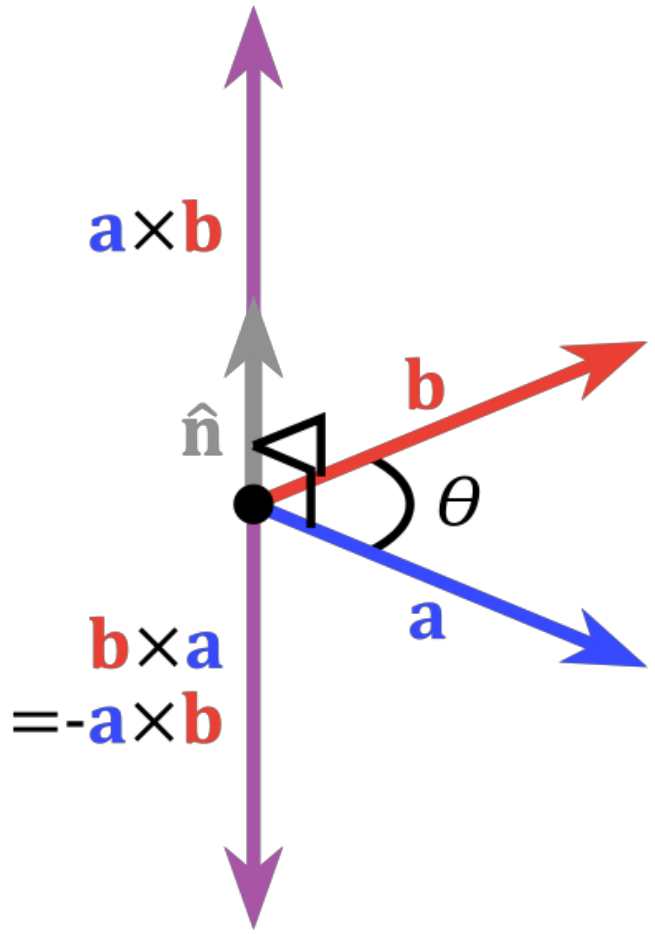


$$\hat{a} = \frac{1}{|a|} a$$

Vector Dot & Cross Product

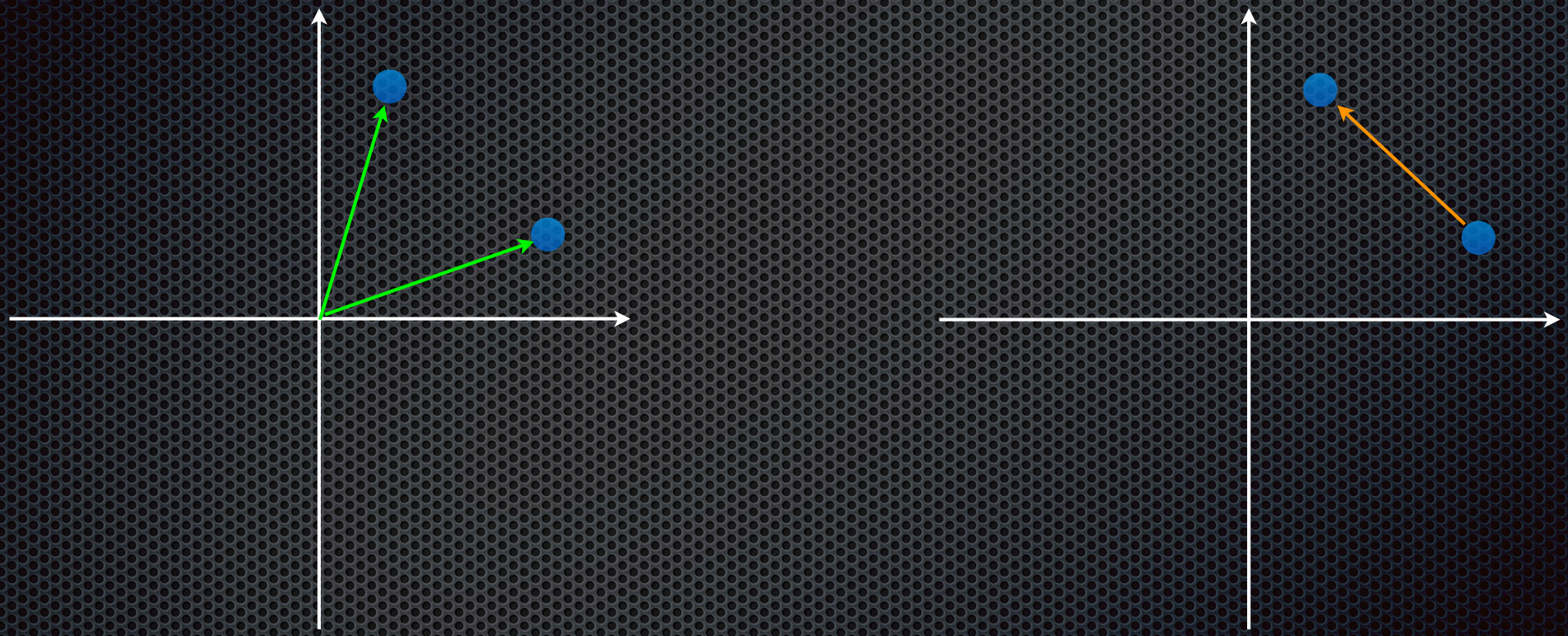


$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}|\,|\mathbf{b}| \cos\theta$$



$$\mathbf{a} \times \mathbf{b} = \|\mathbf{a}\|\,\|\mathbf{b}\| \sin(\theta)\,\mathbf{n}$$

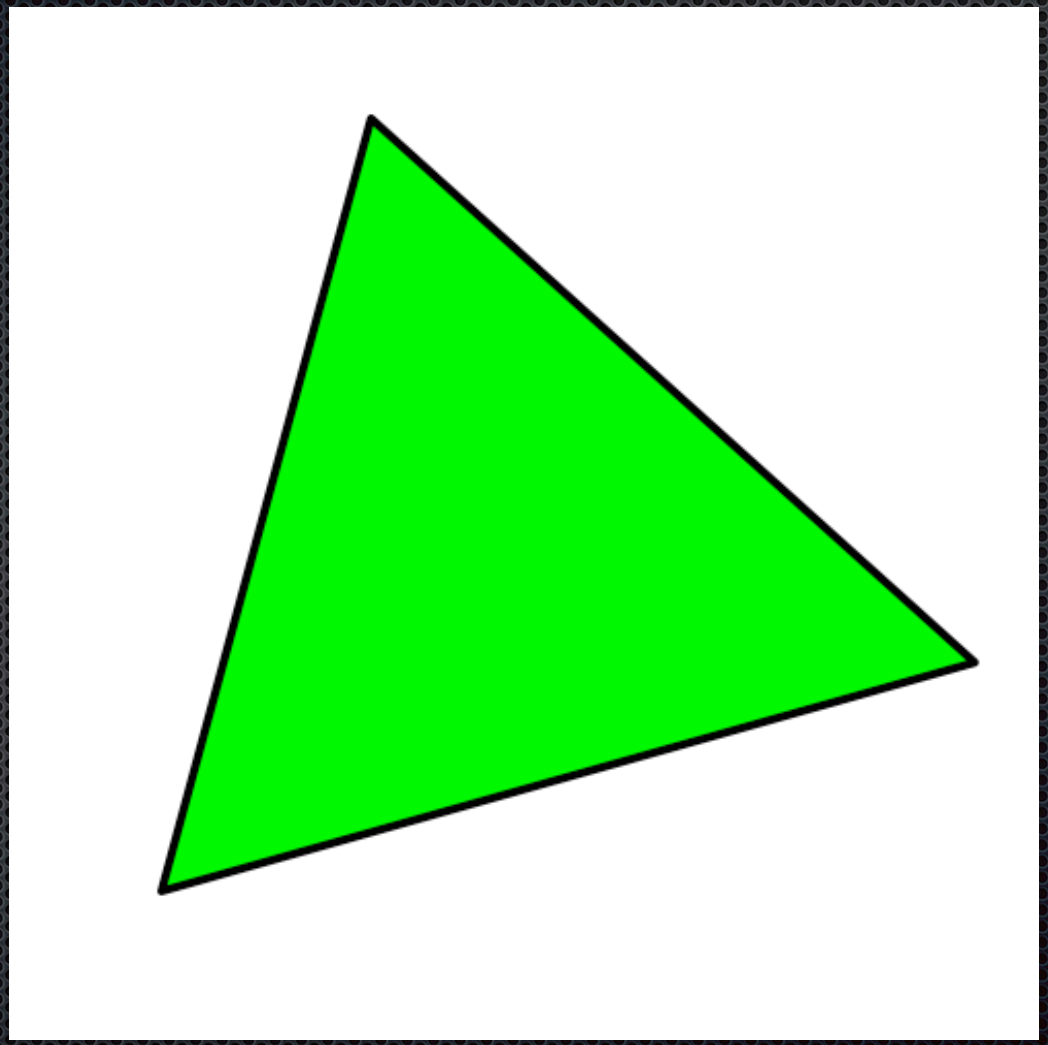
Point and Free Vectors



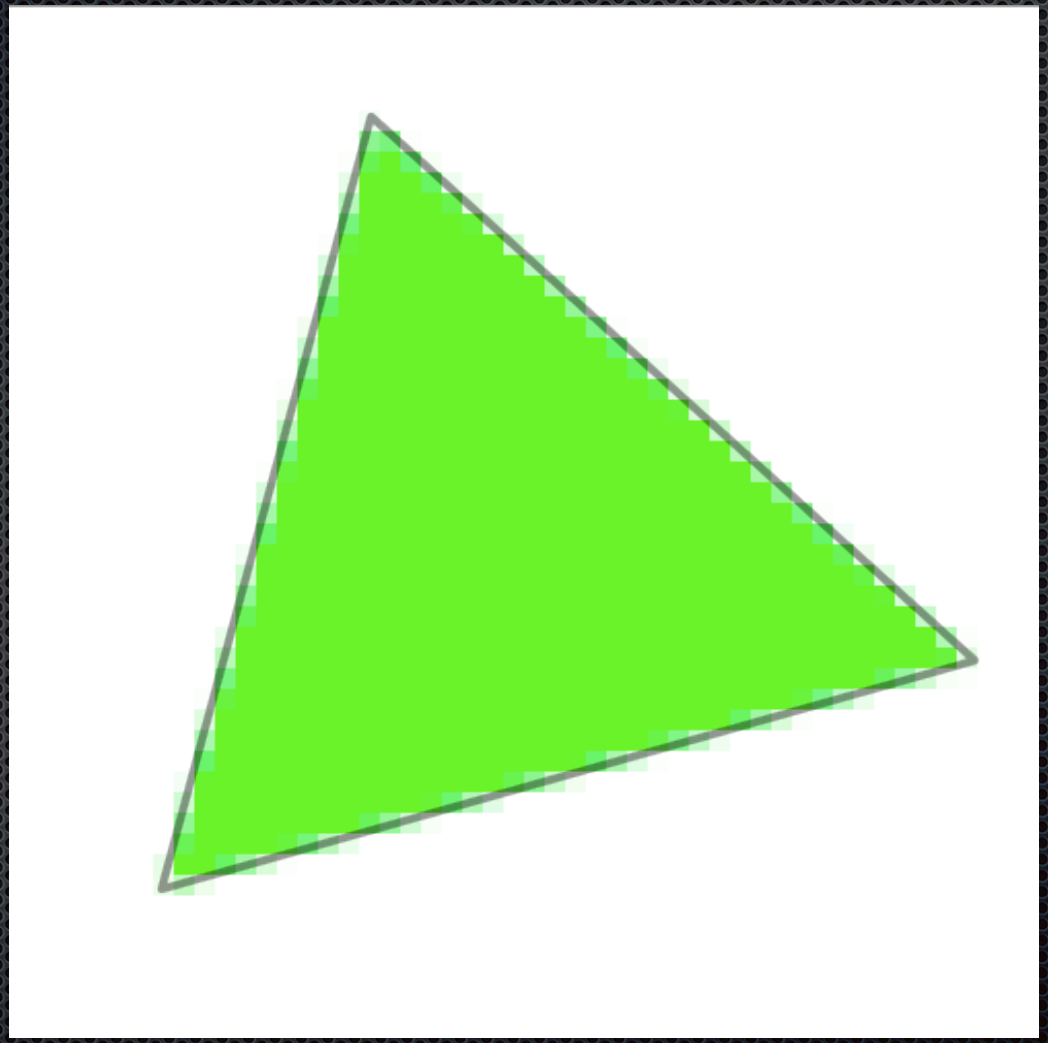
Point

Free

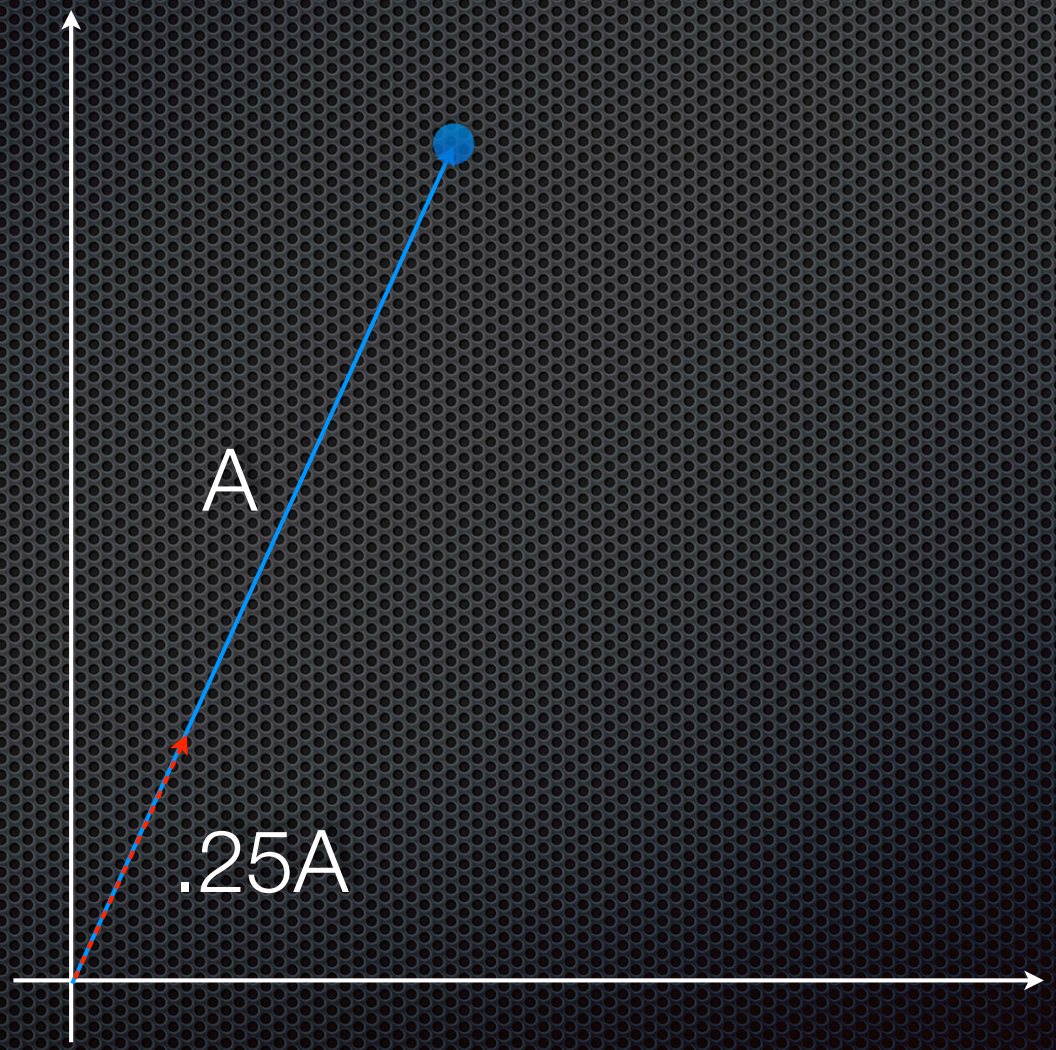
Rasterization



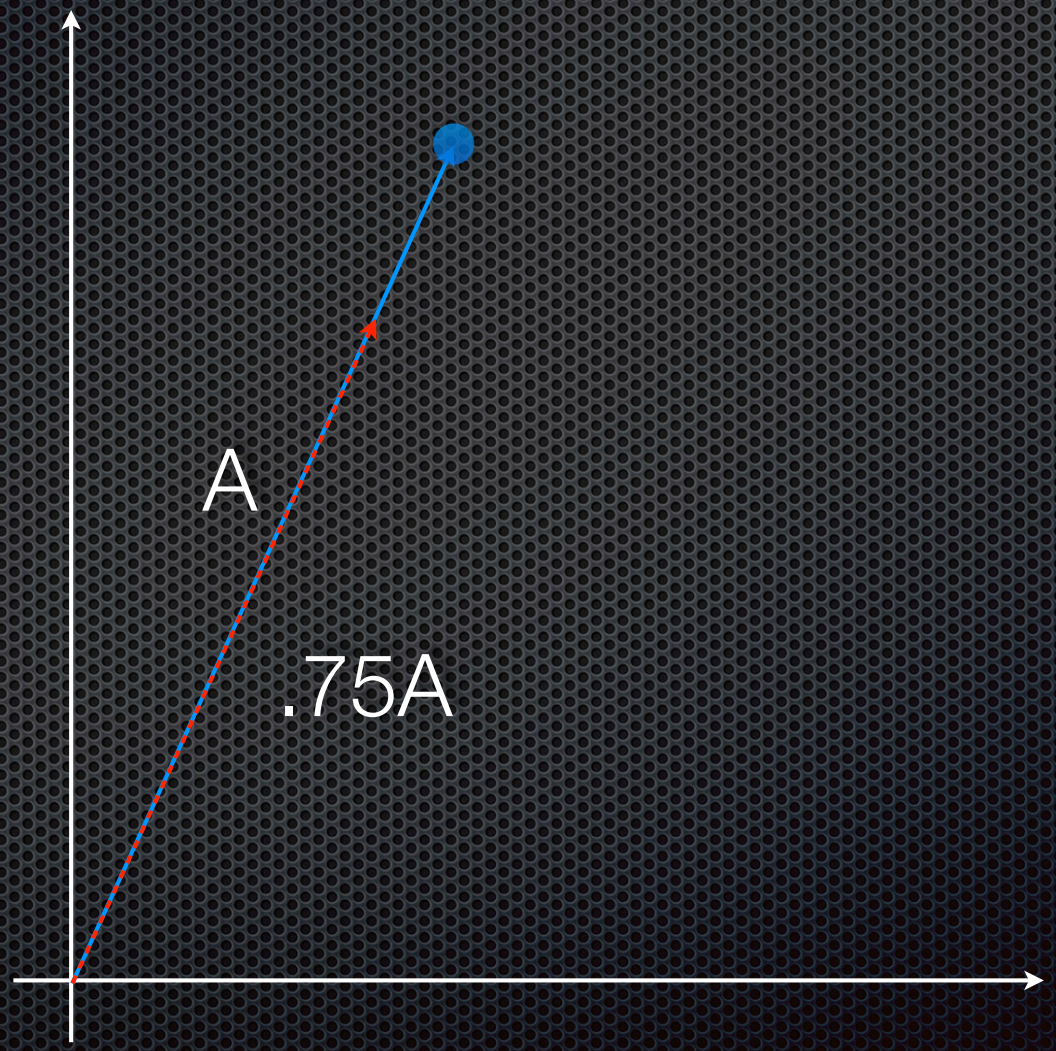
Rasterization



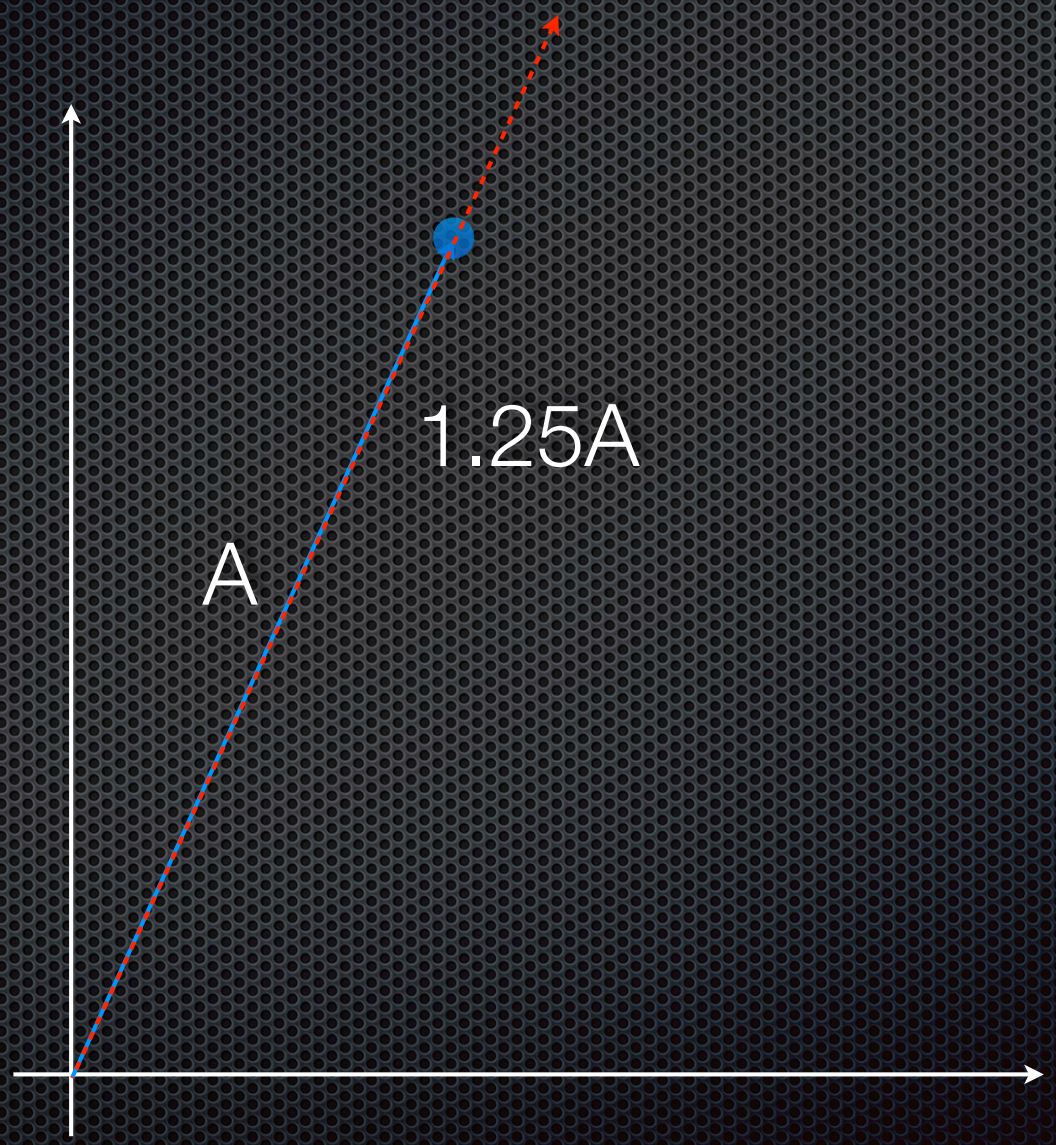
An Interesting Idea



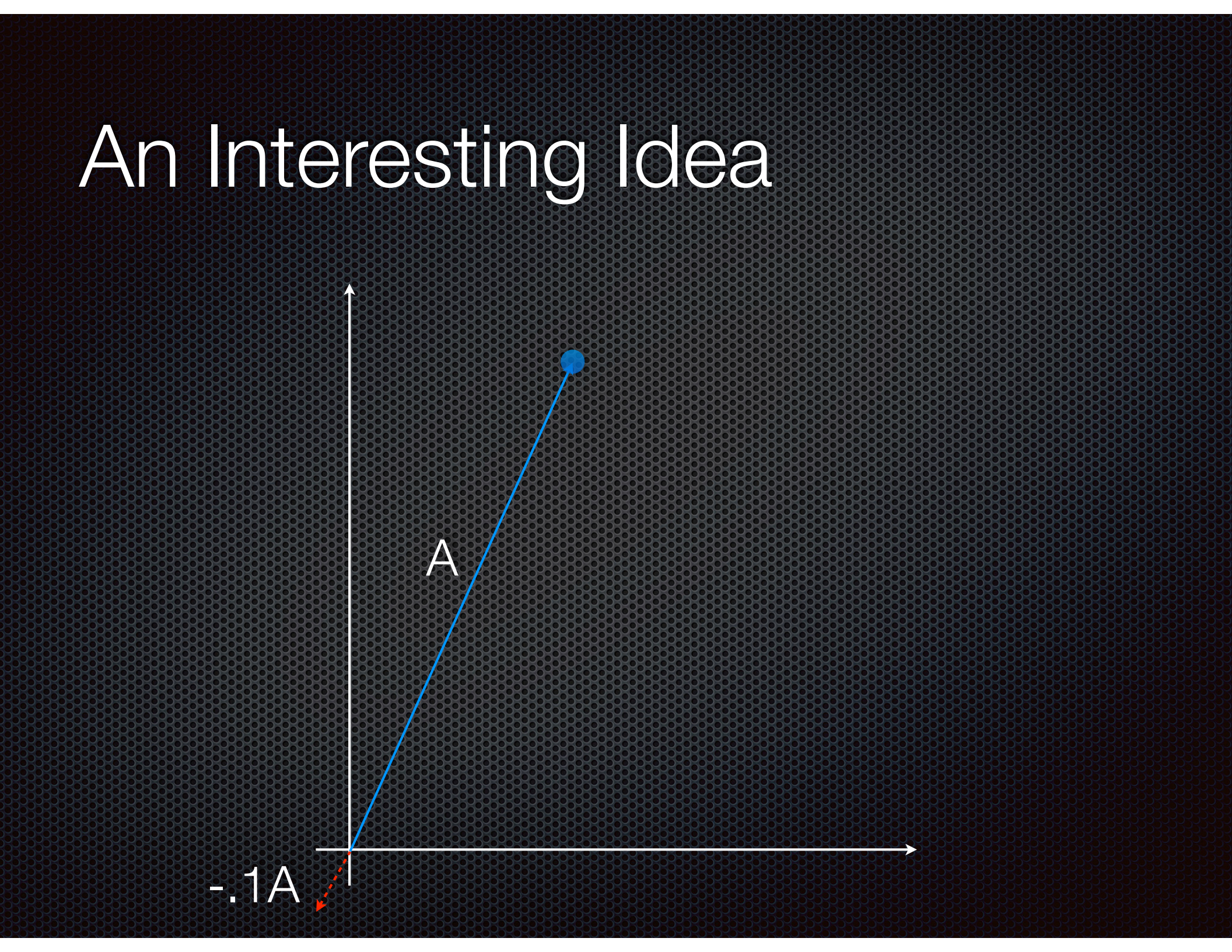
An Interesting Idea



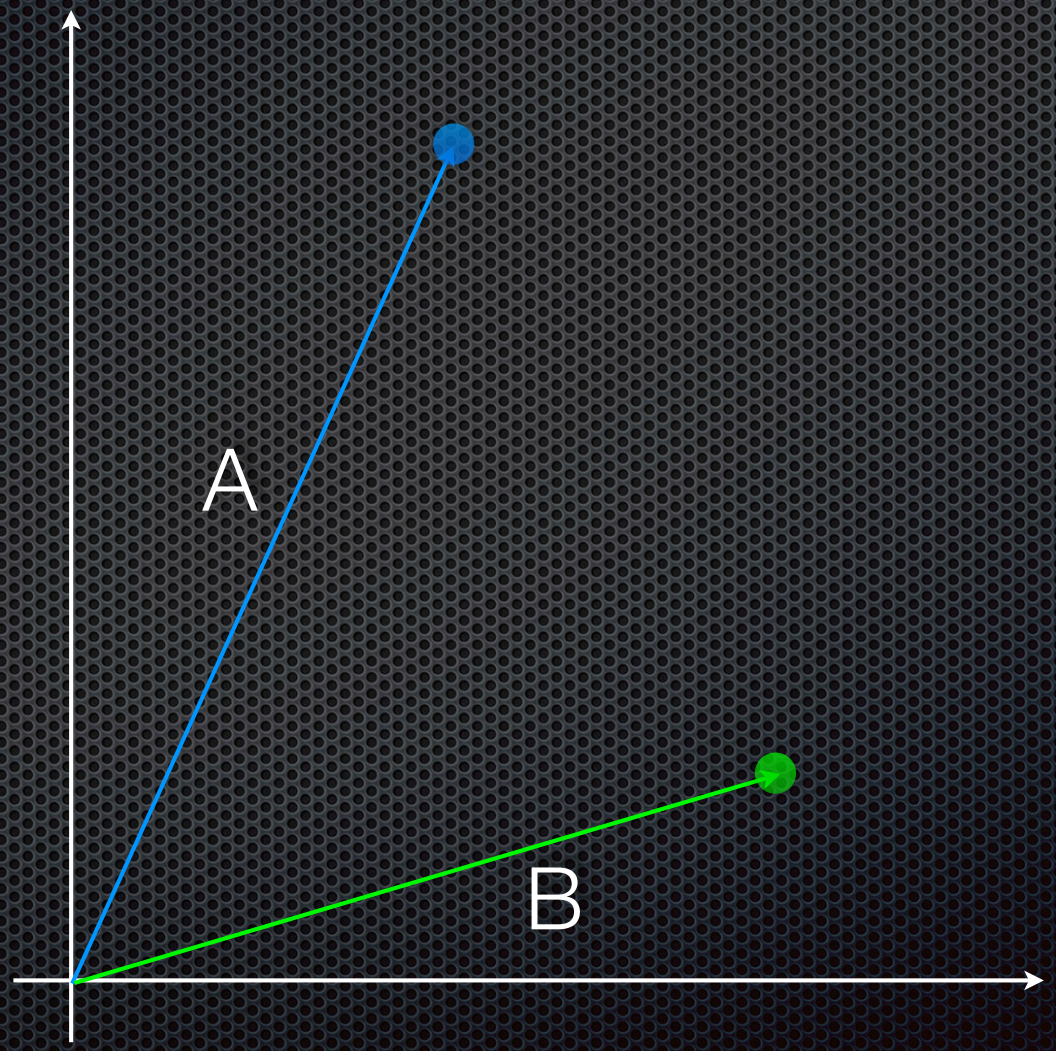
An Interesting Idea



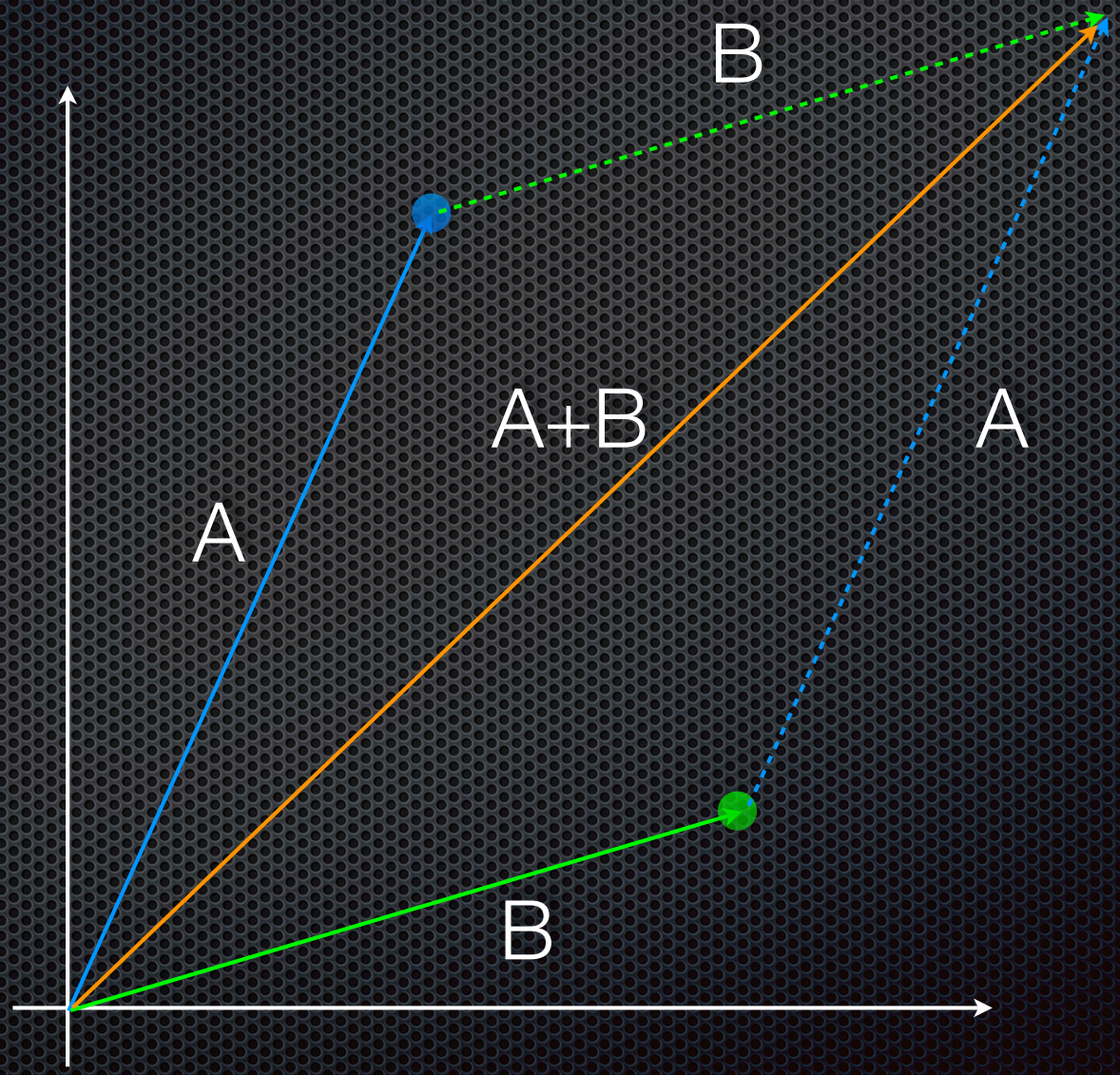
An Interesting Idea



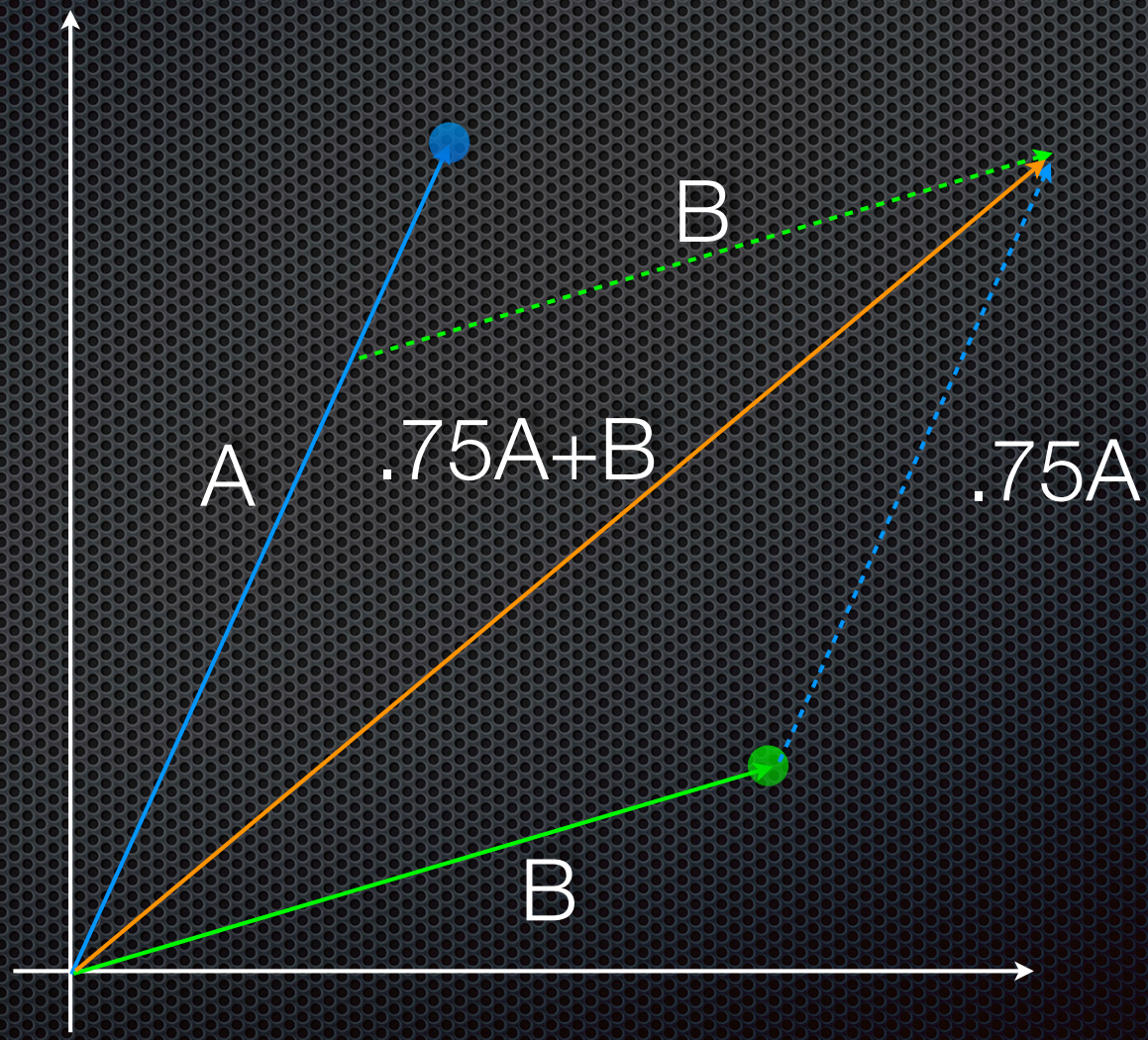
An Interesting Idea



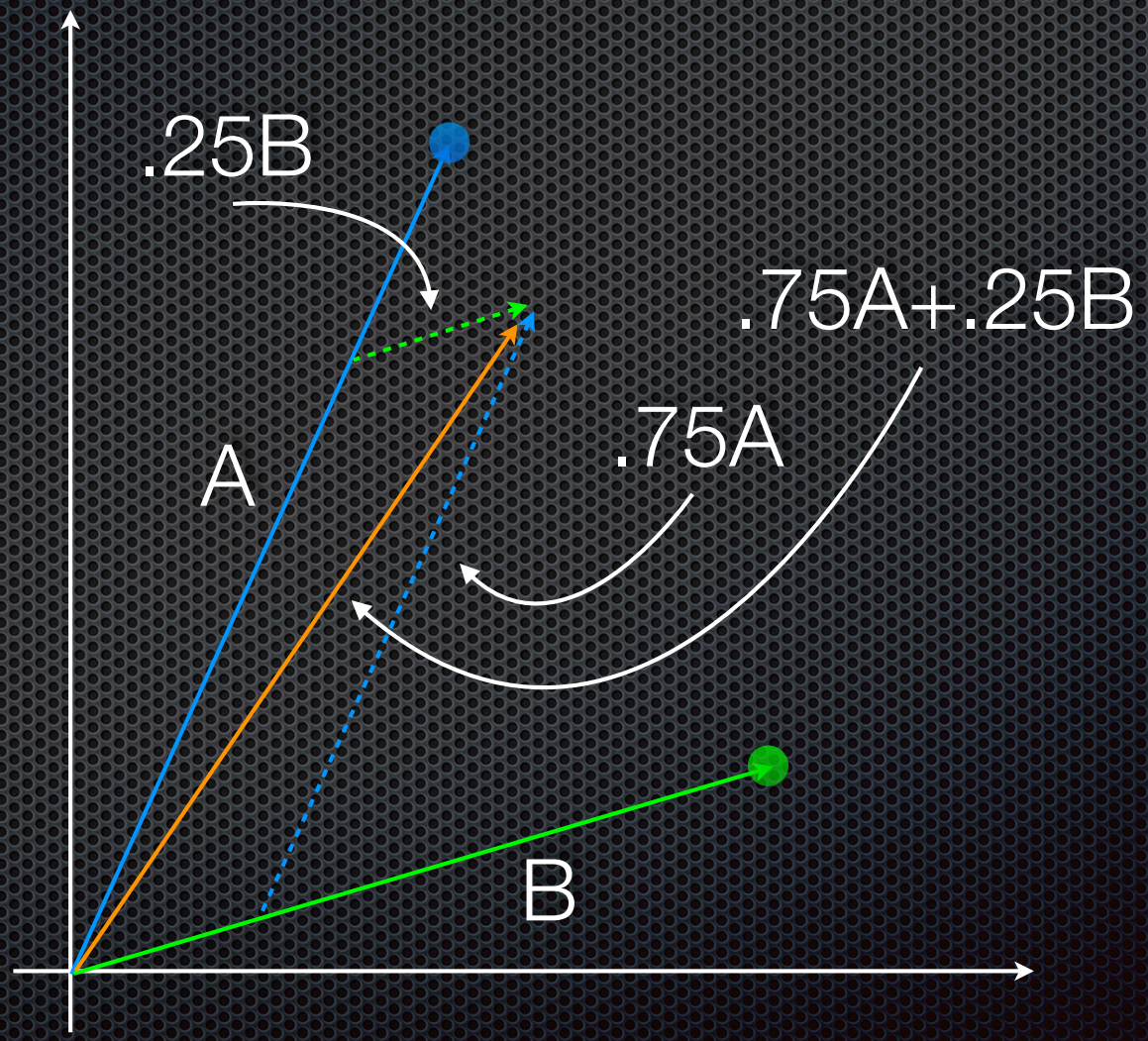
An Interesting Idea



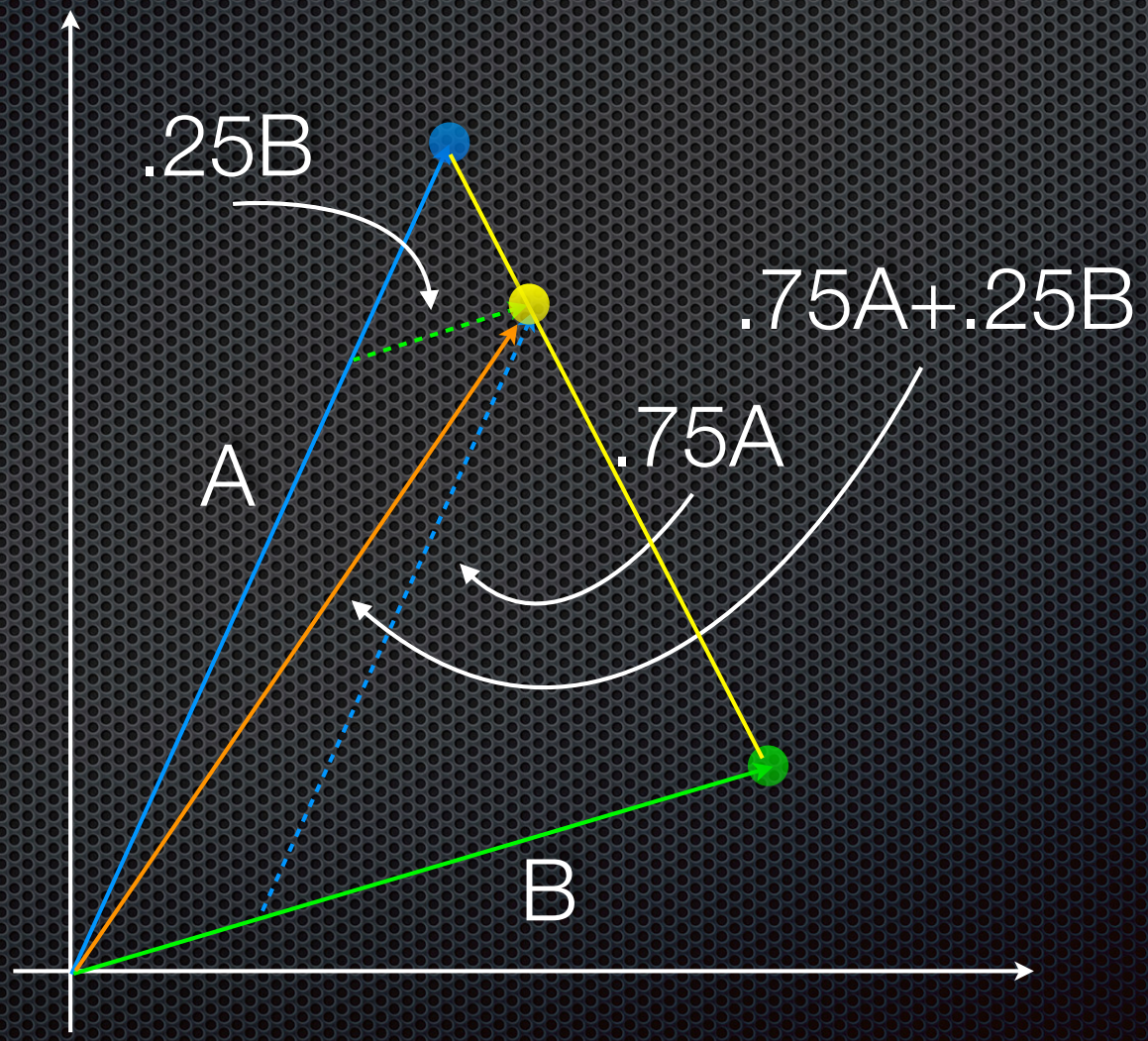
Linear Combination



Linear Combination

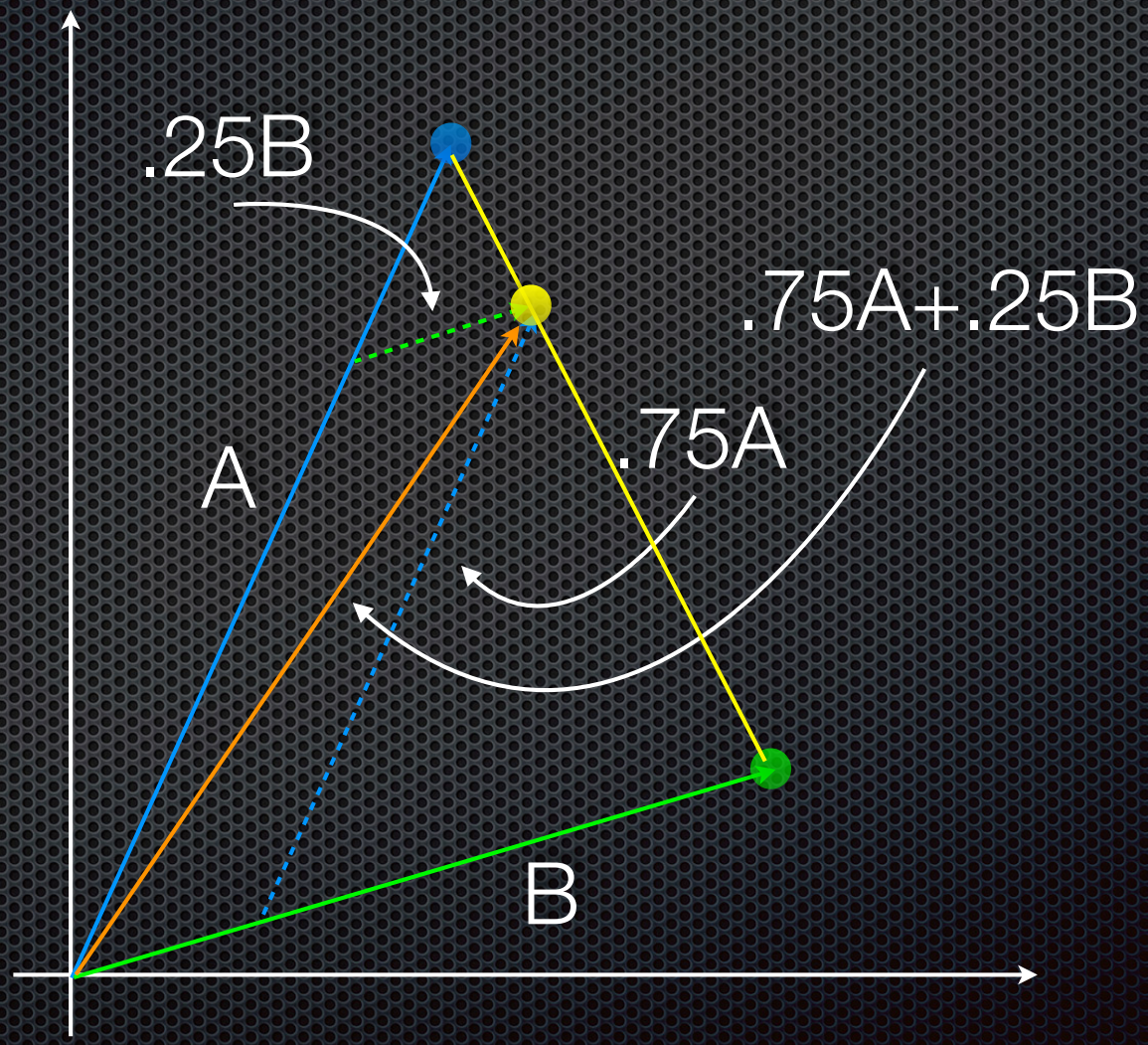


Linear Combination



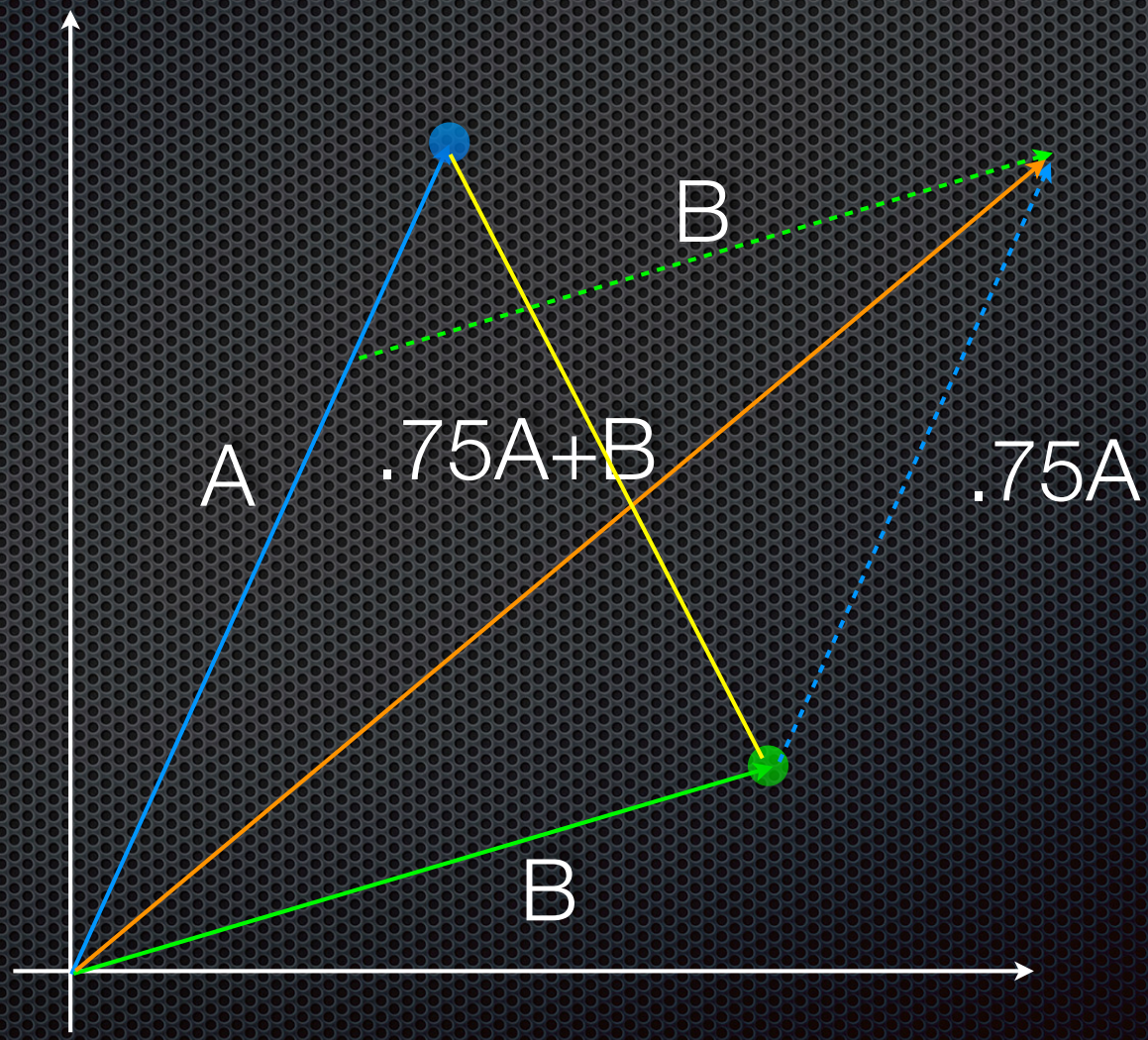
Linear Combination or Blend

.75 + .25 = 1



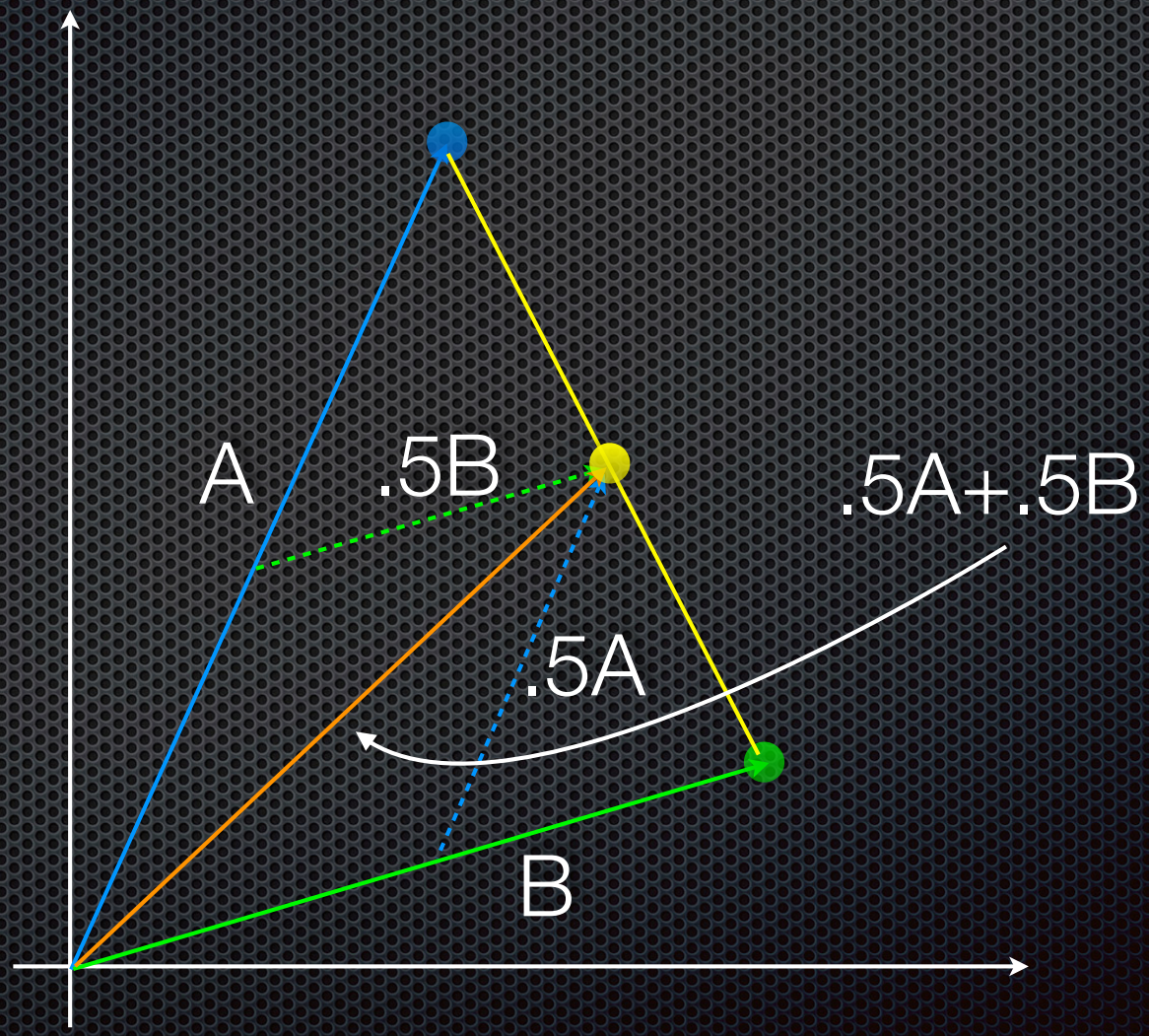
Linear Combination

.75 + 1 ≠ 1

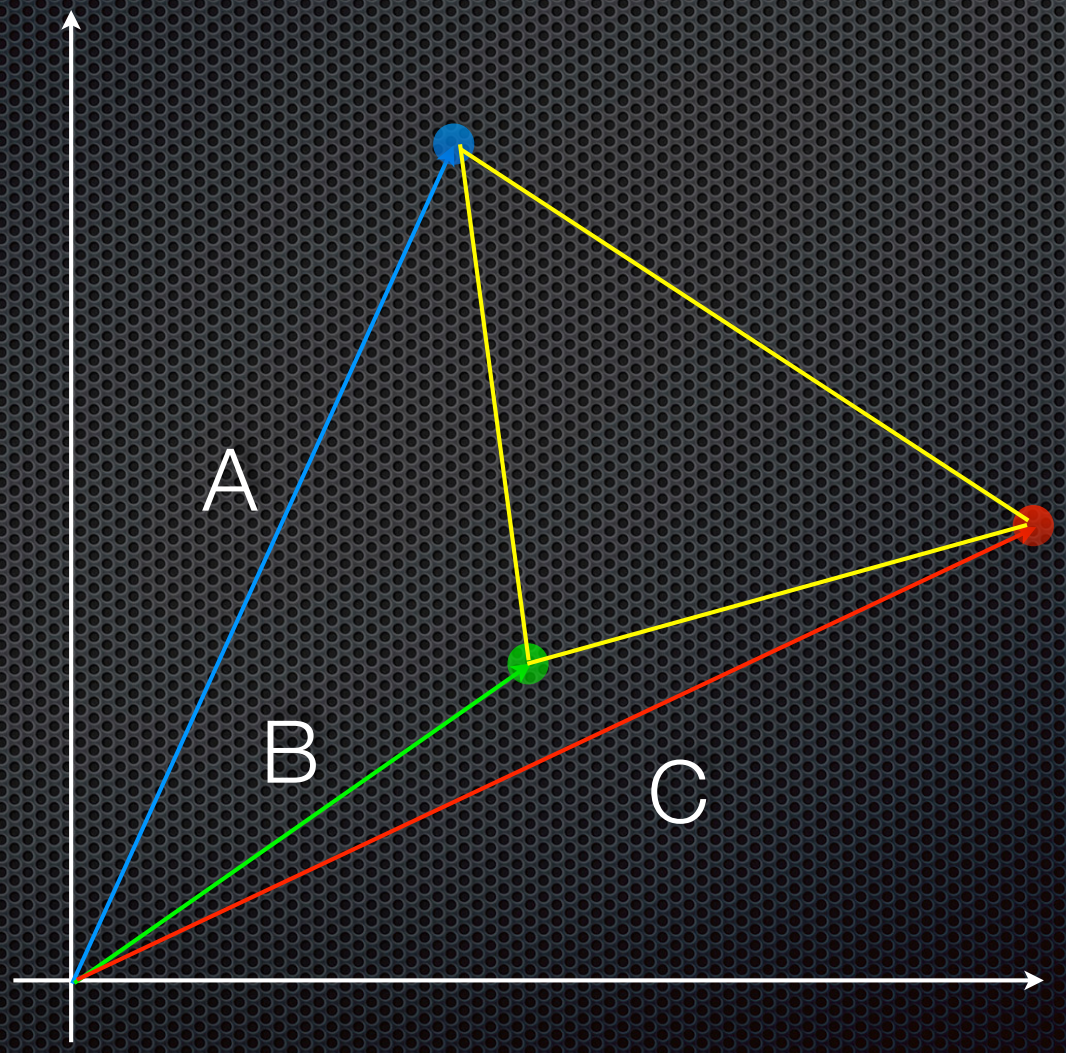


Linear Combination or Blend

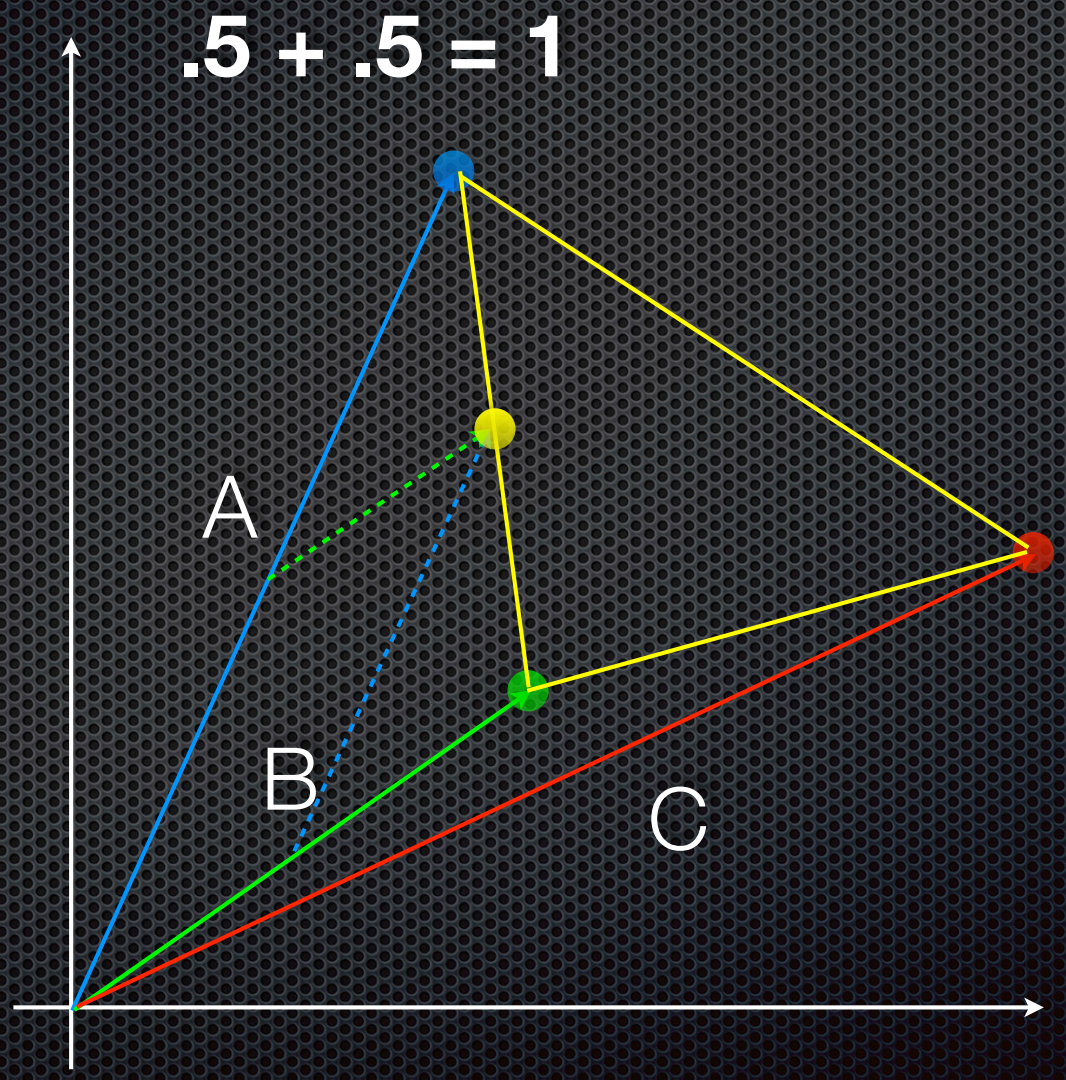
.5 + .5 = 1



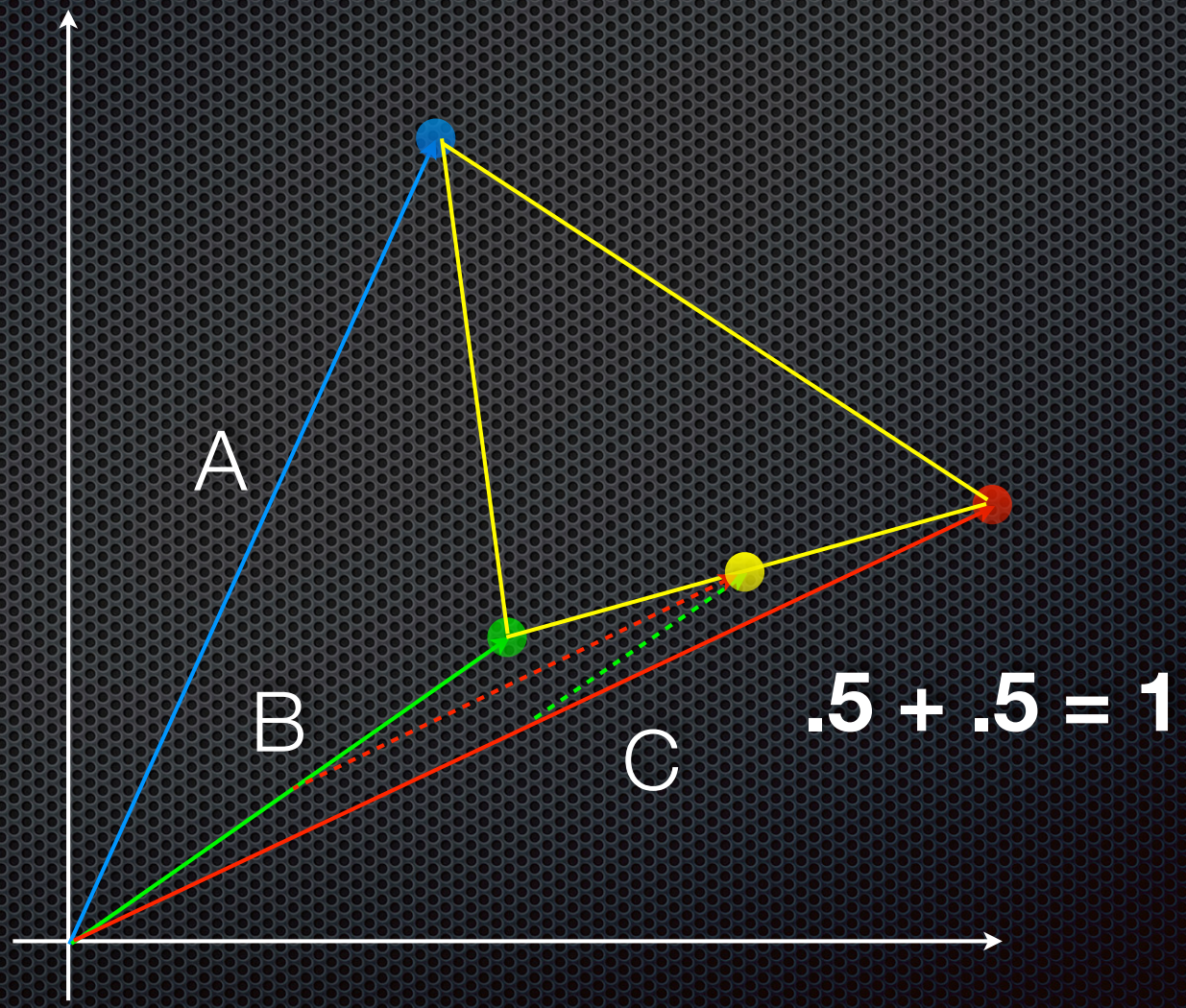
Triangle



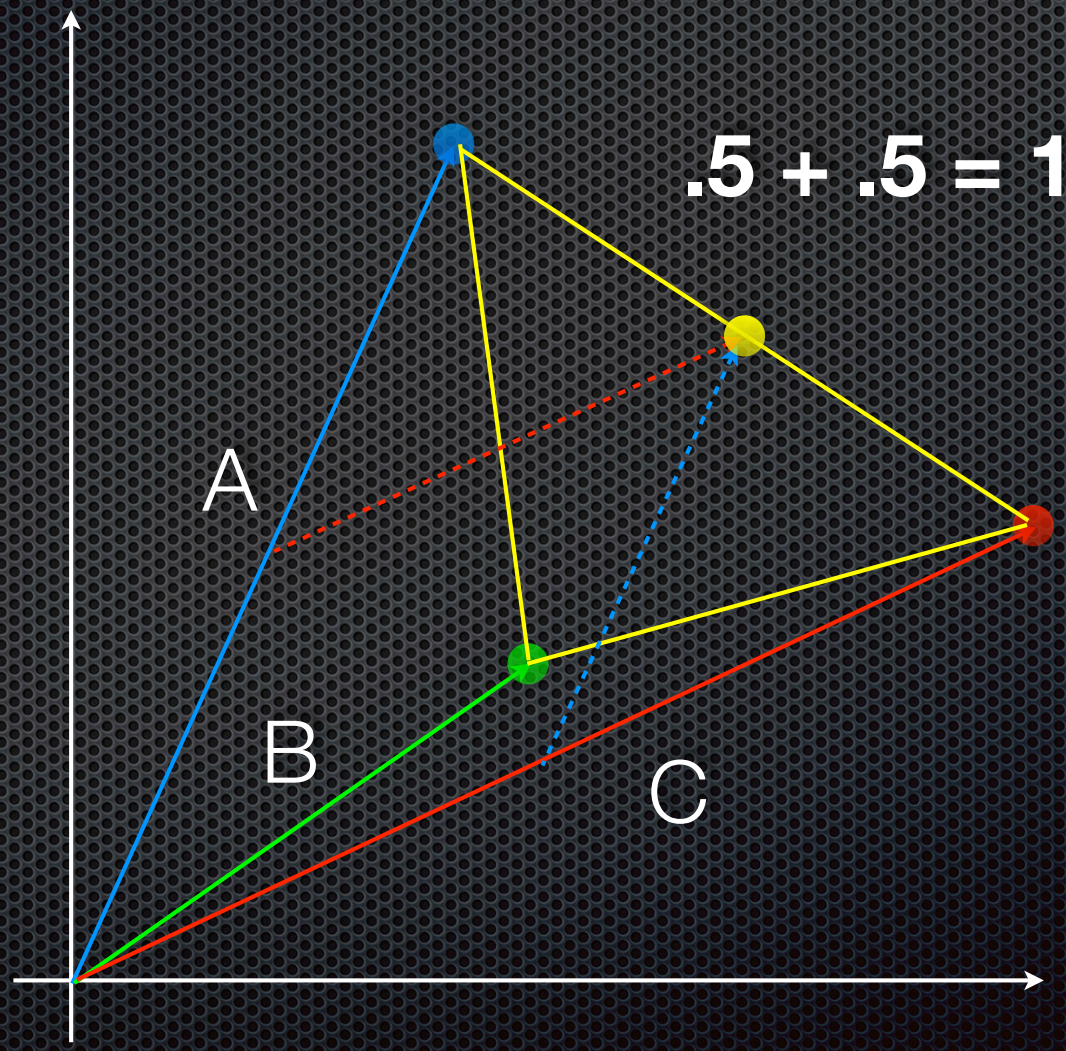
Triangle



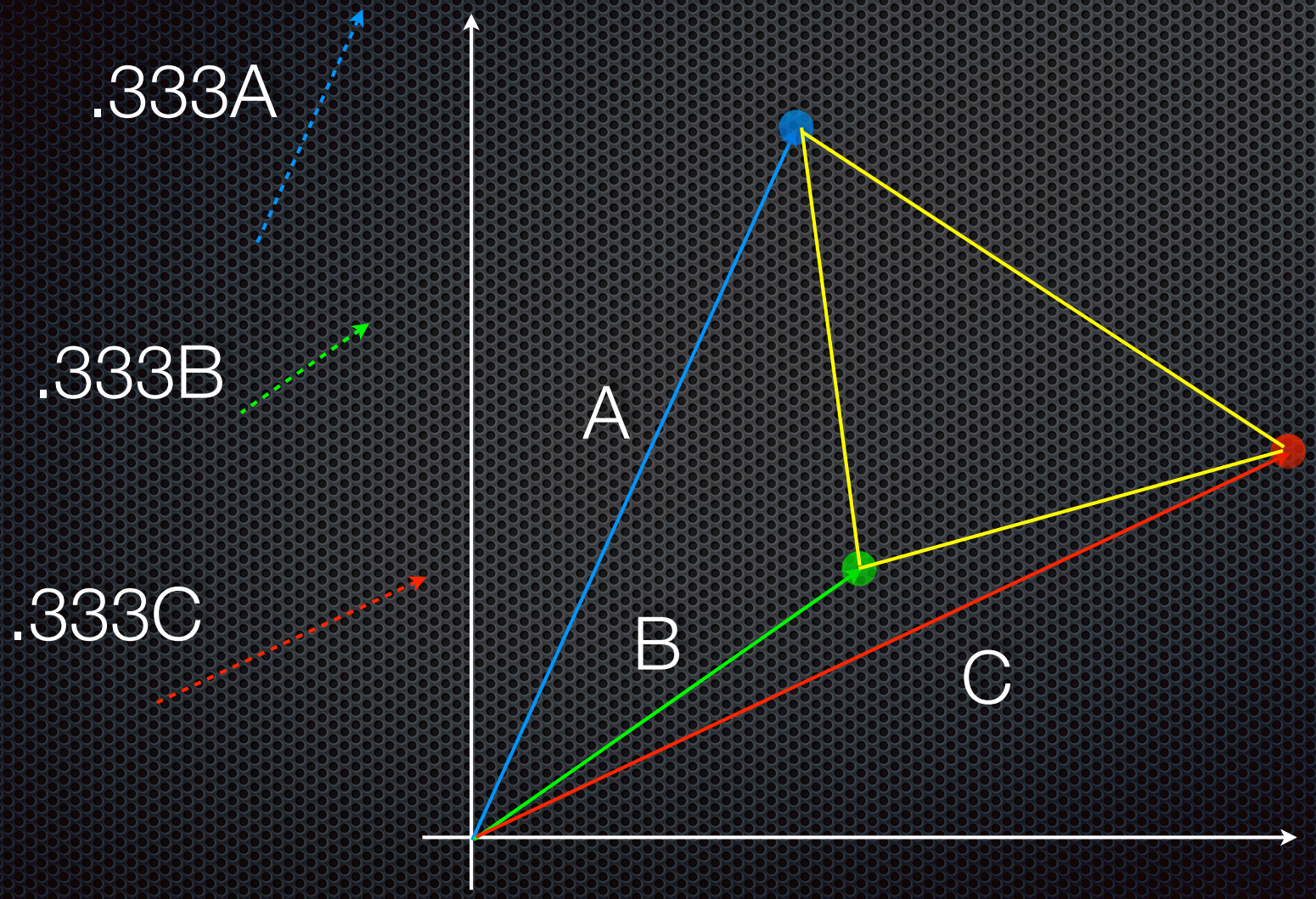
Triangle



Triangle

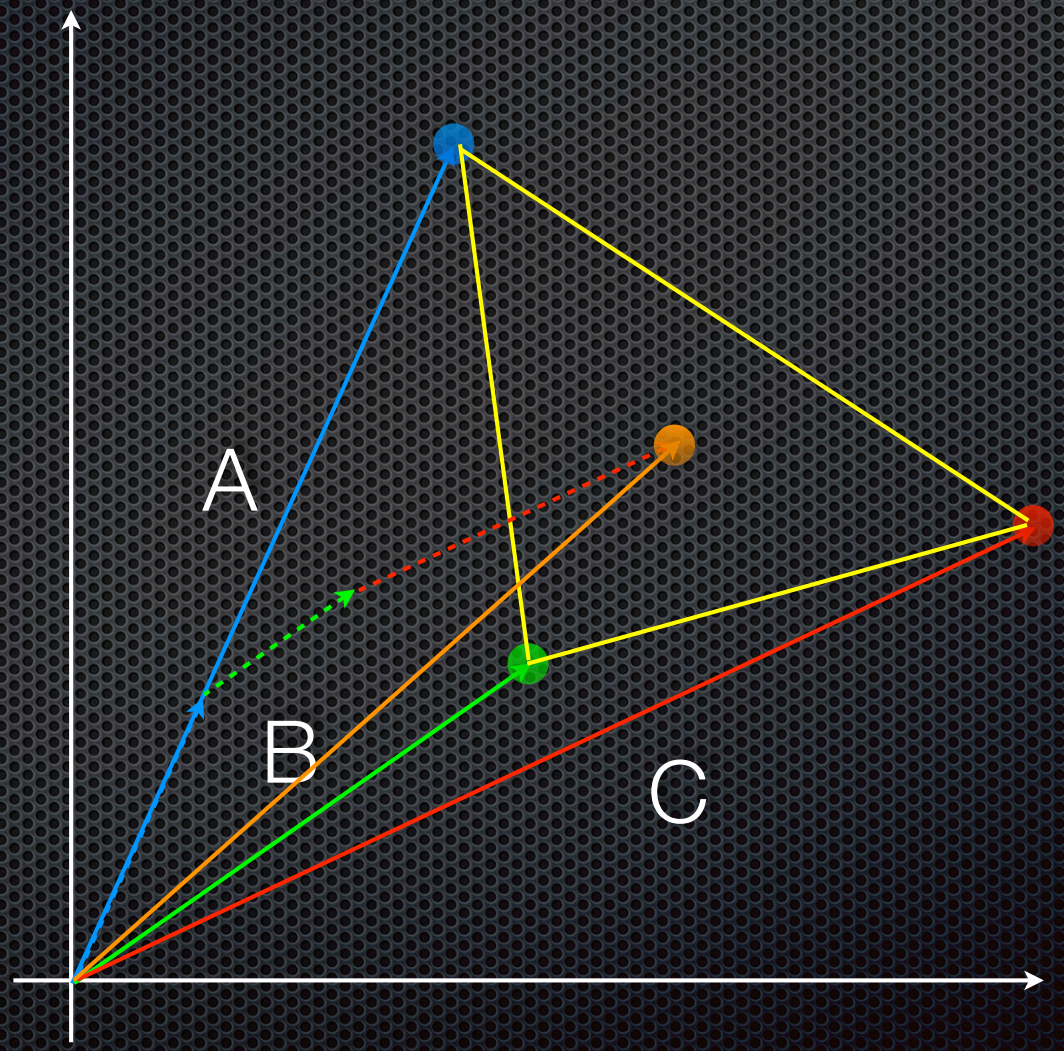


Triangle

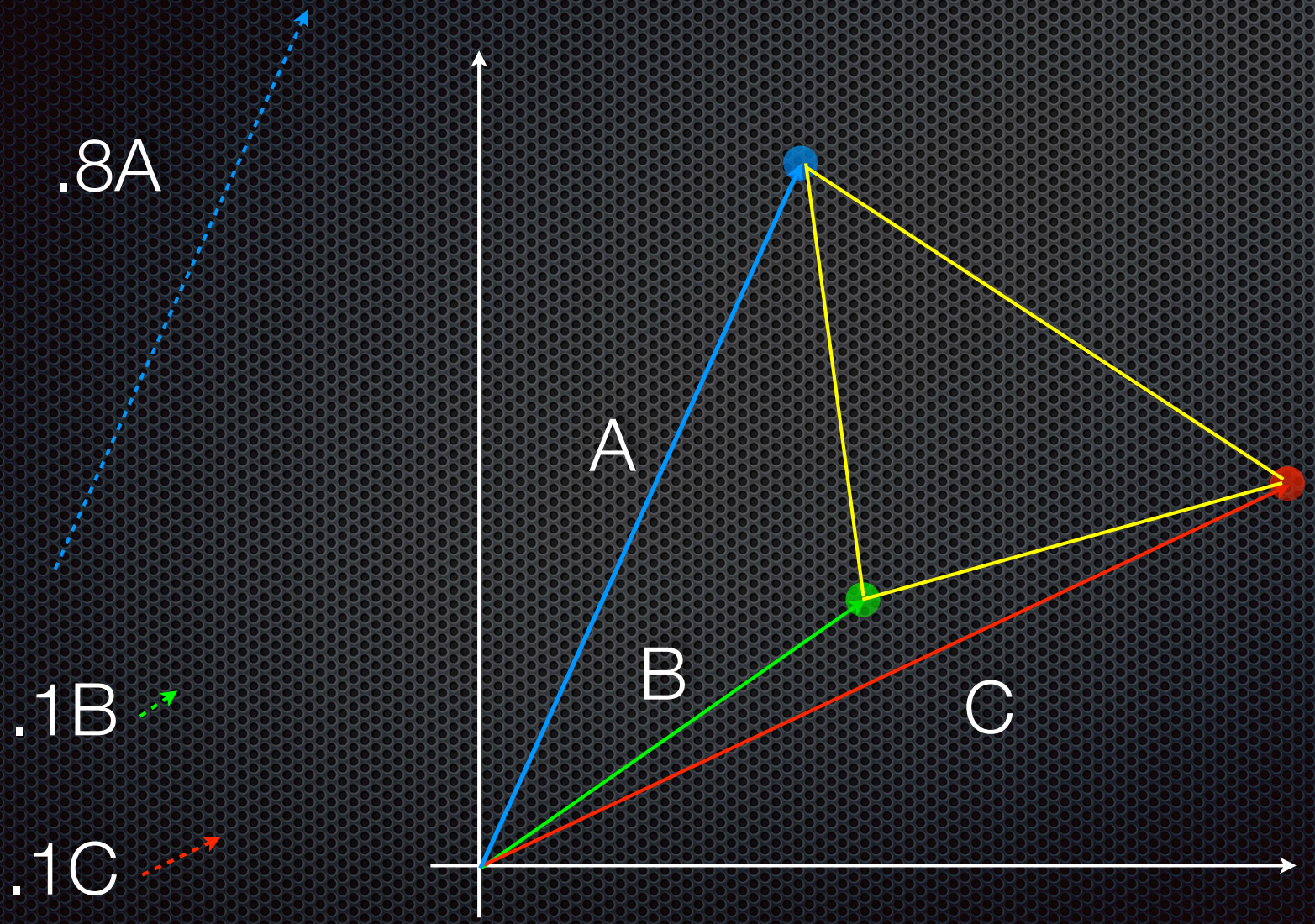


Triangle

.333 + .333 + .333 = 1

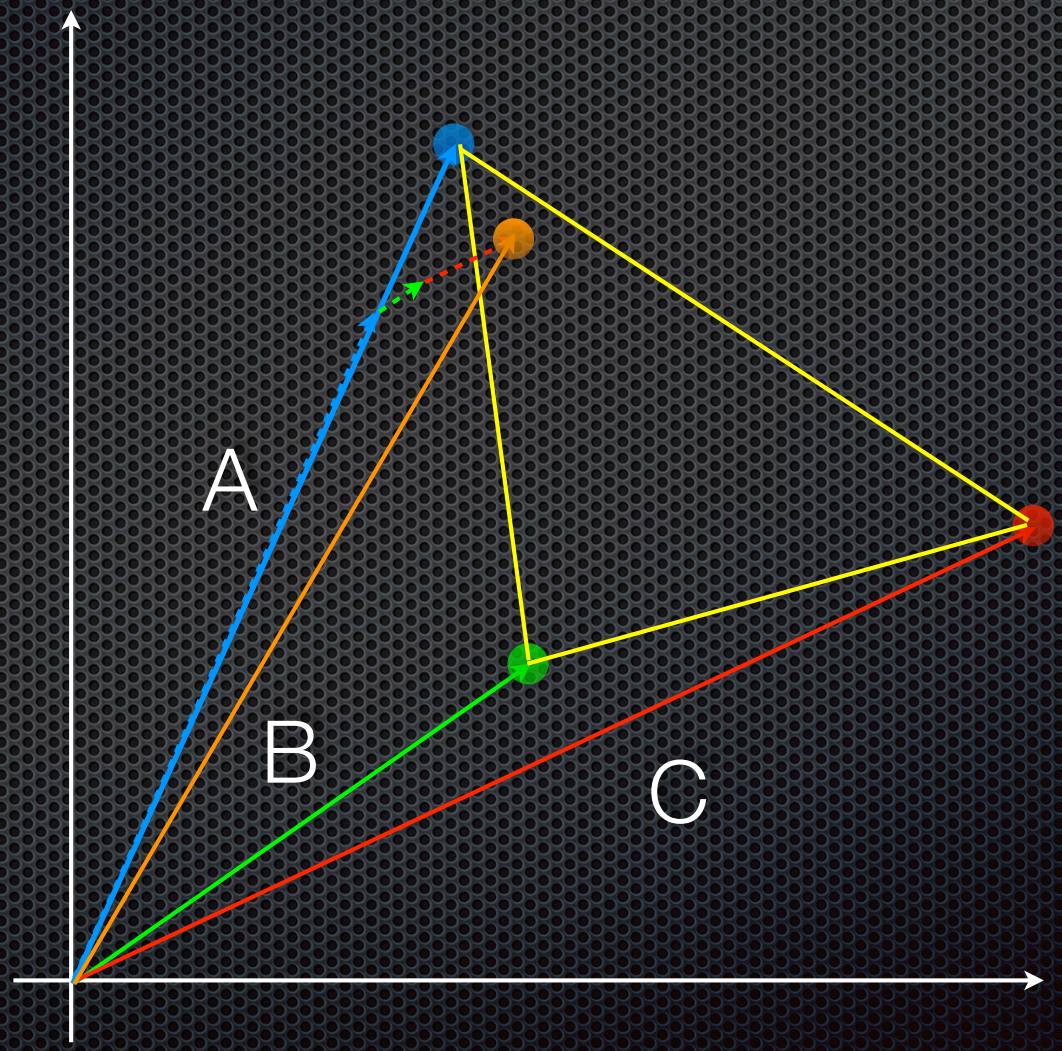


Triangle

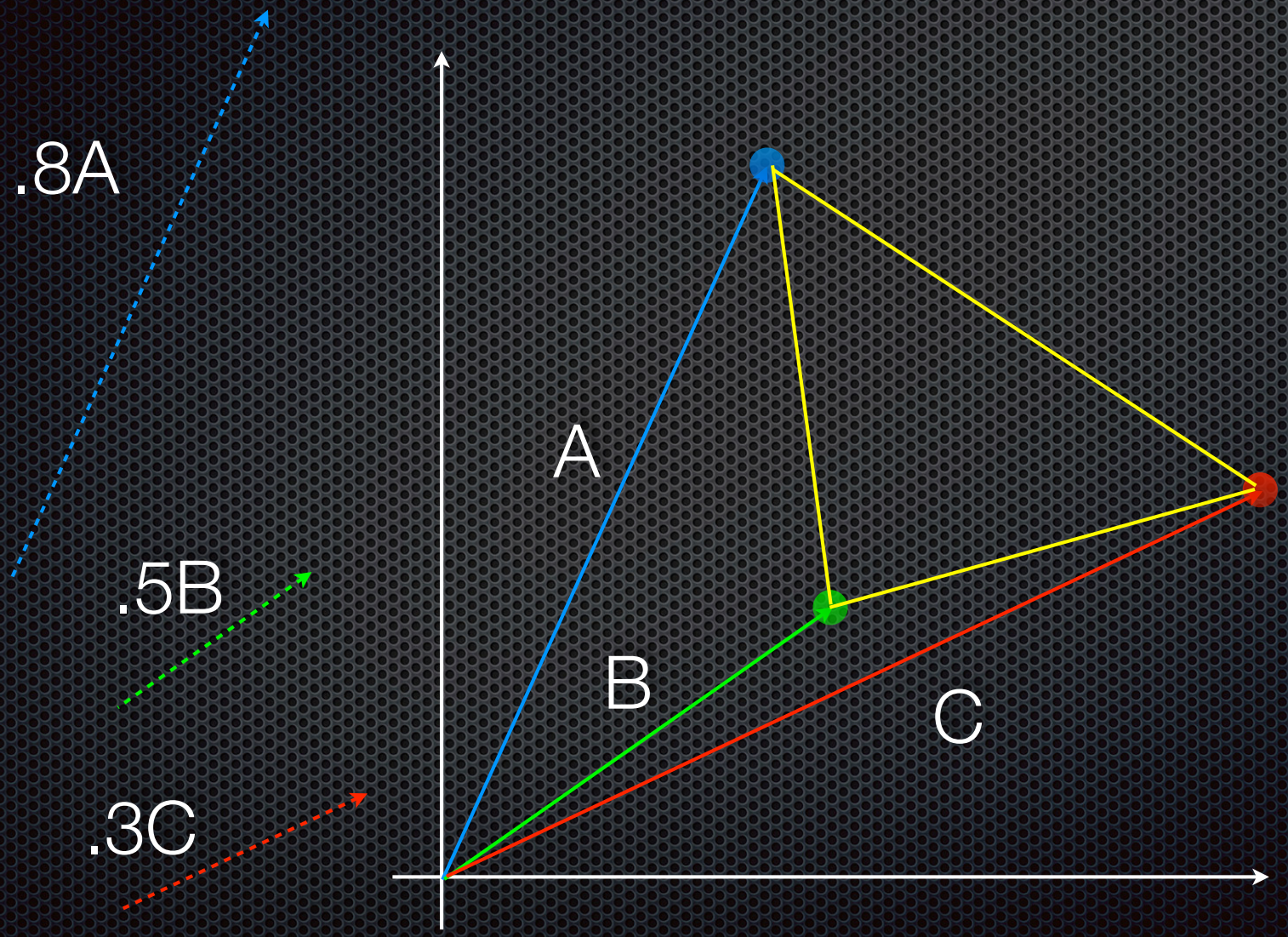


Triangle

.8 + .1 + .1 = 1

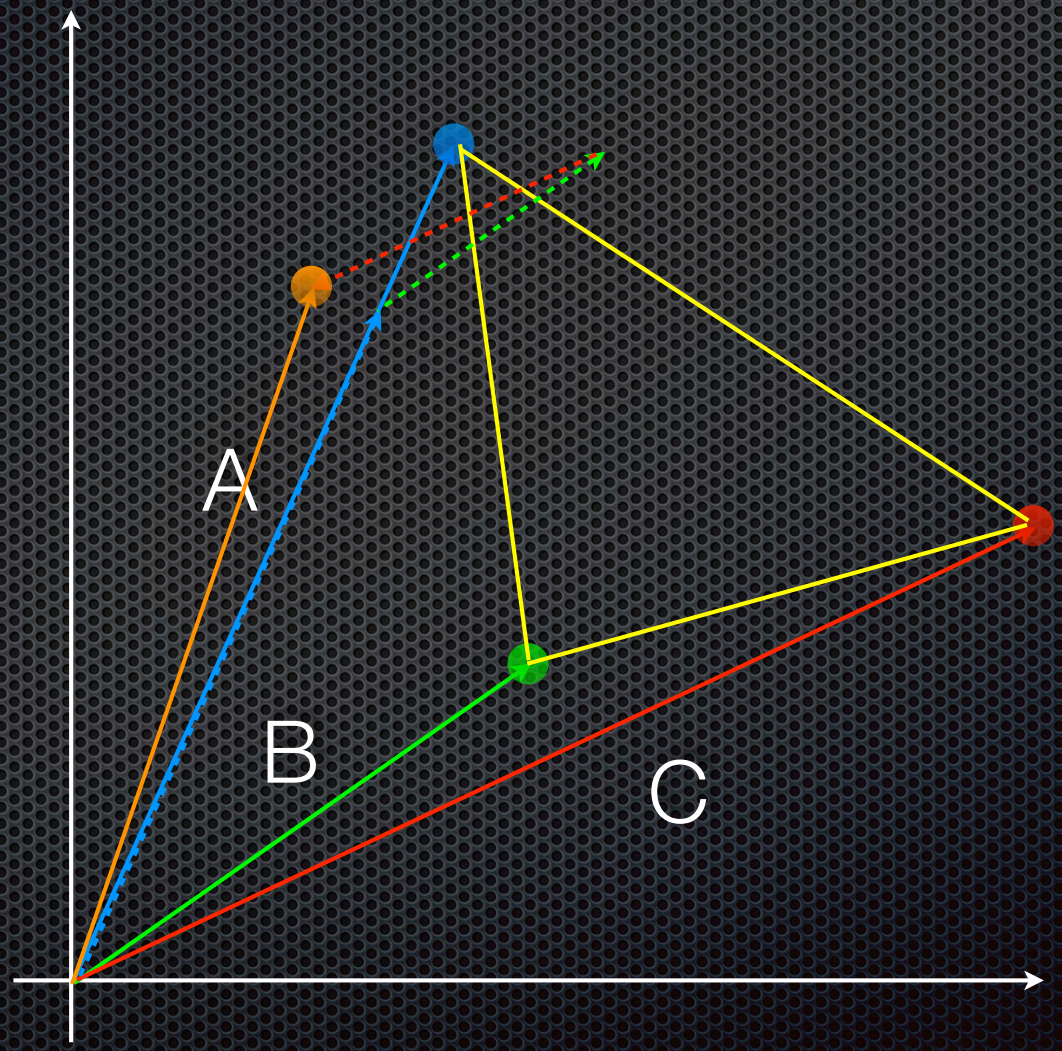


Triangle



Triangle

.8 + .5 - .3 = 1

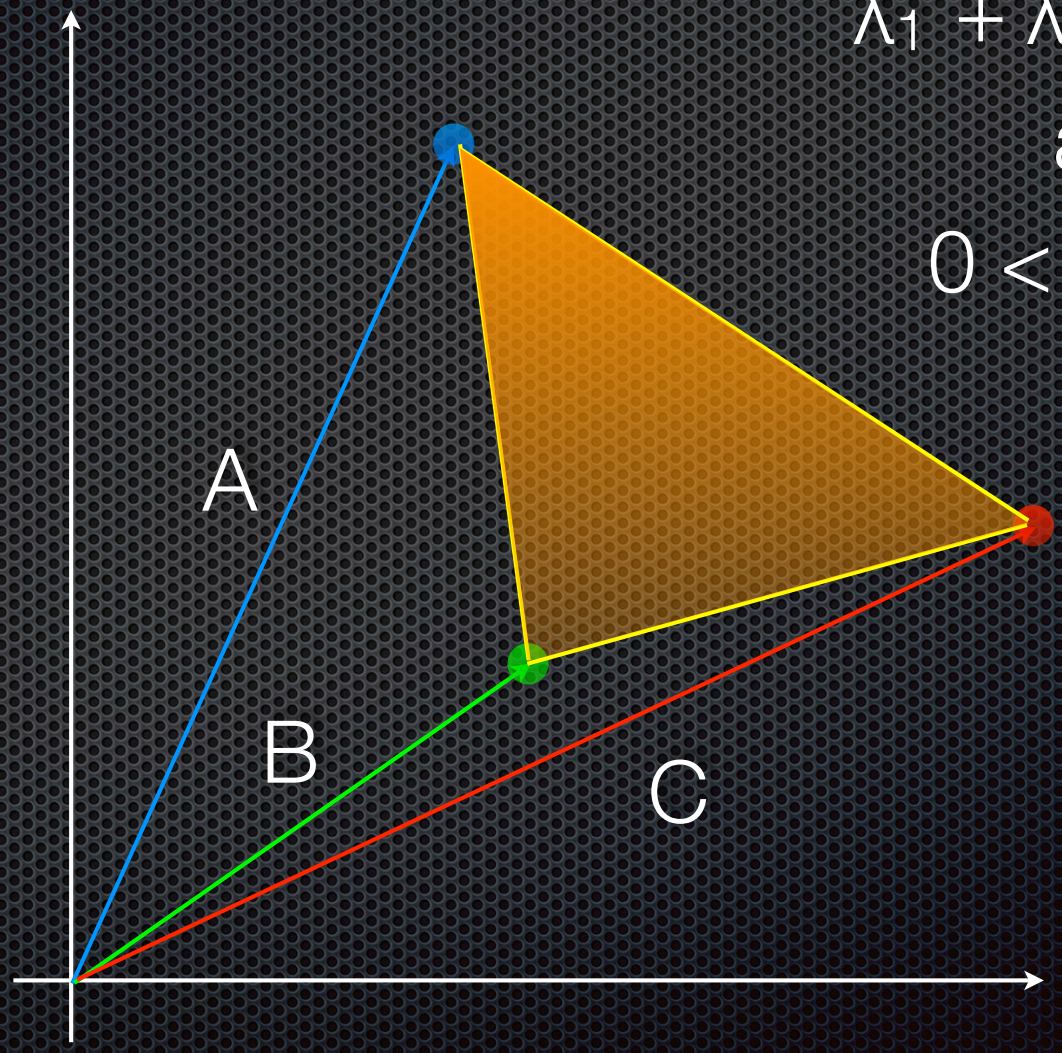


Barycentric Coordinates

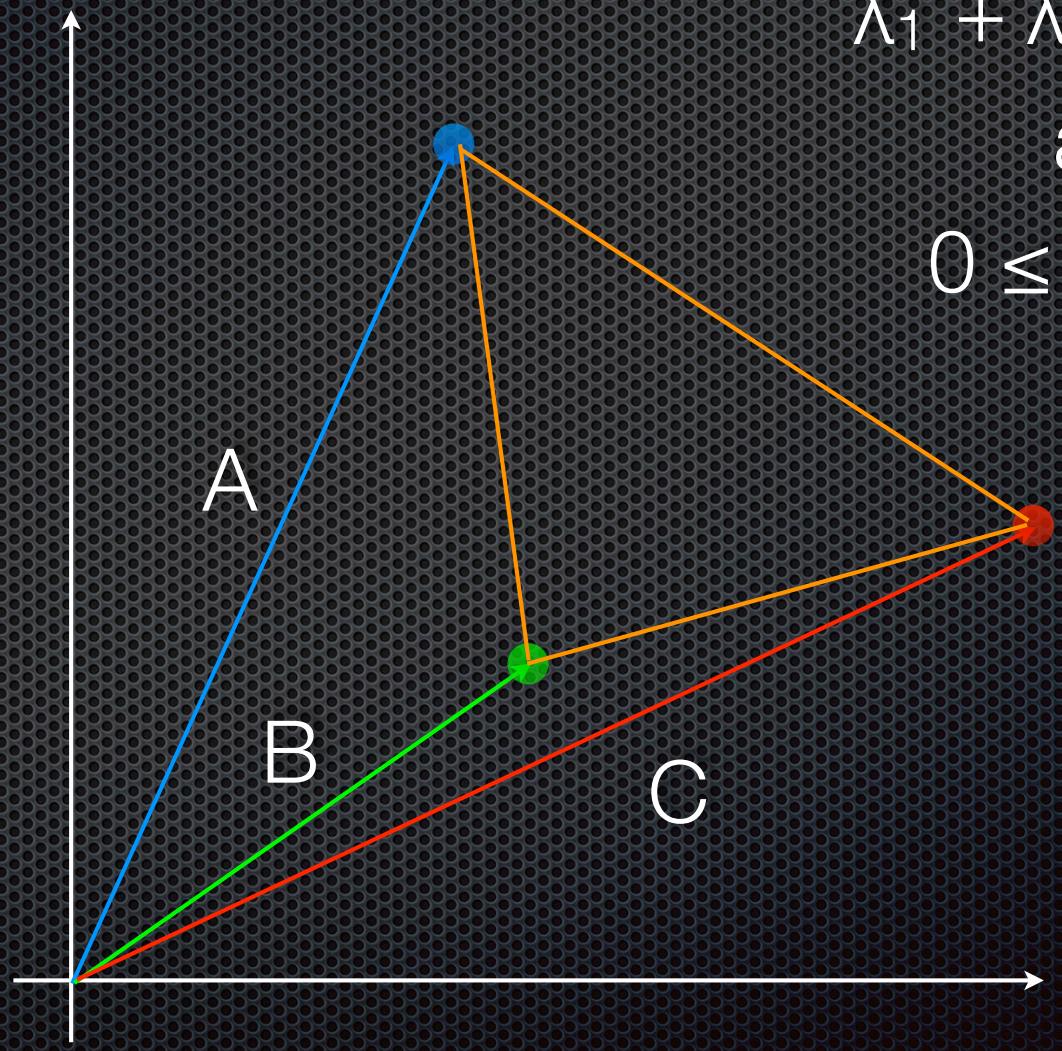
$\lambda_1 + \lambda_2 + \lambda_3 = 1$

and

$0 < \lambda_x < 1$



Barycentric Coordinates



$\lambda_1 + \lambda_2 + \lambda_3 = 1$

and

$0 \leq \lambda_x \leq 1$

and

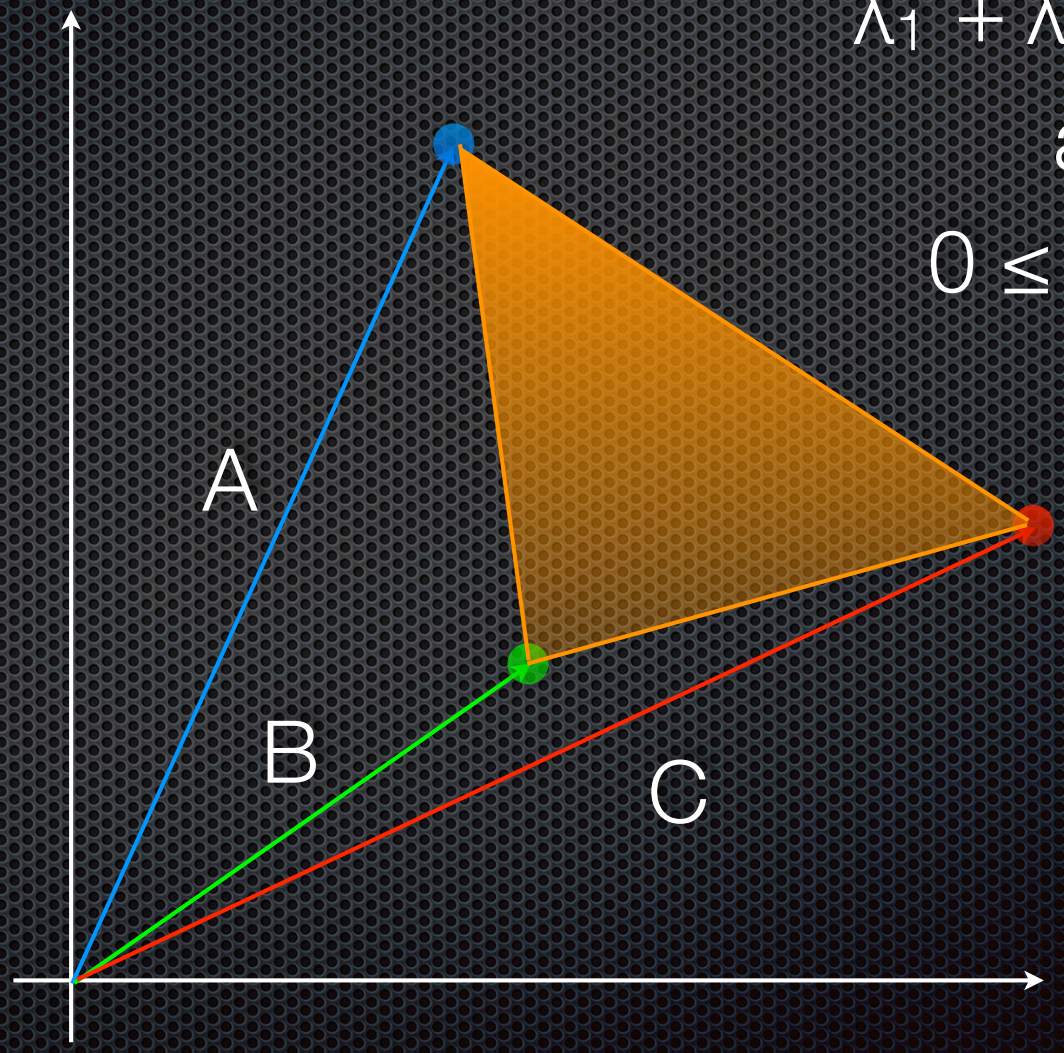
1 or 2 $\lambda_x = 0$

Barycentric Coordinates

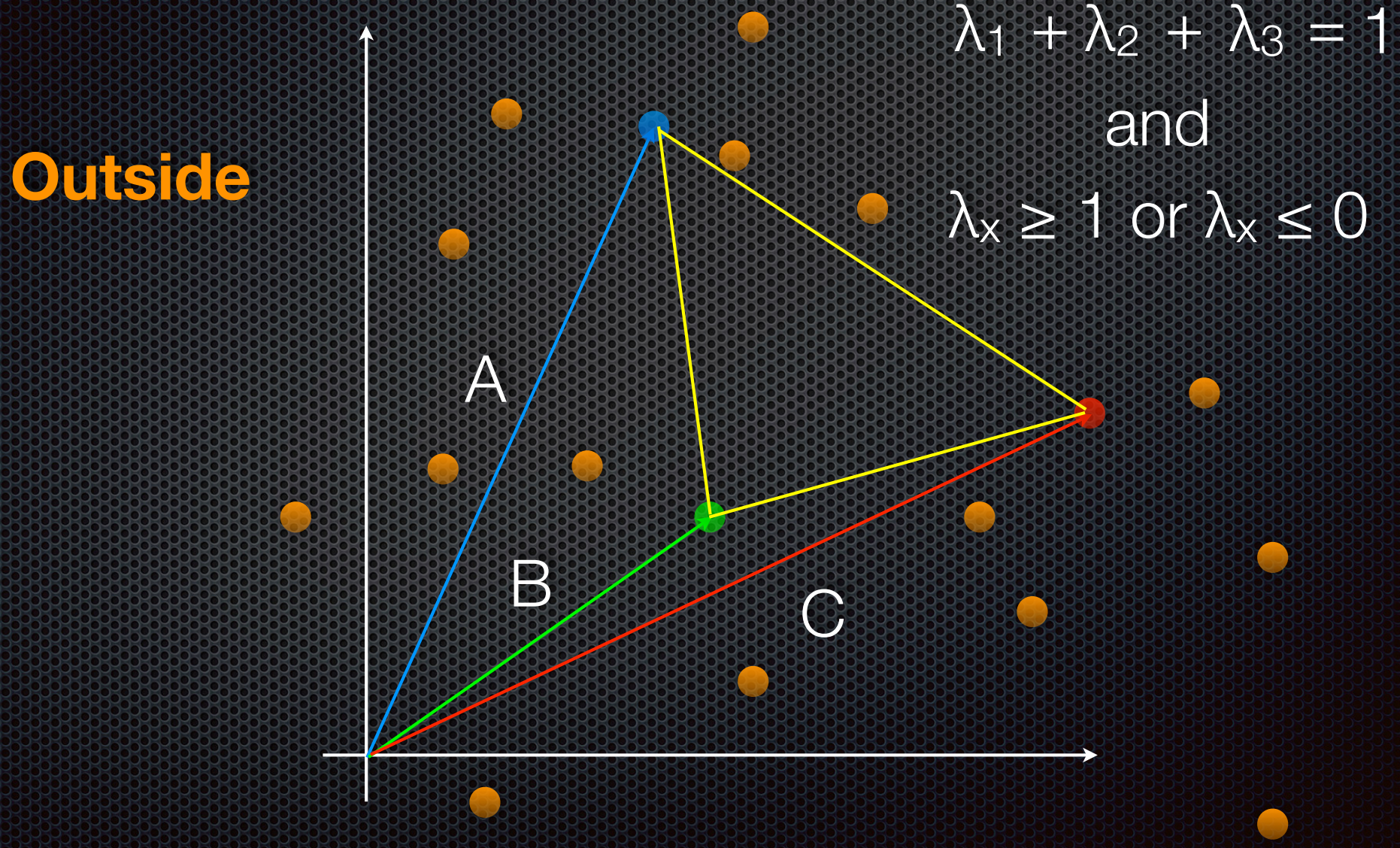
$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

and

$$0 \leq \lambda_x \leq 1$$



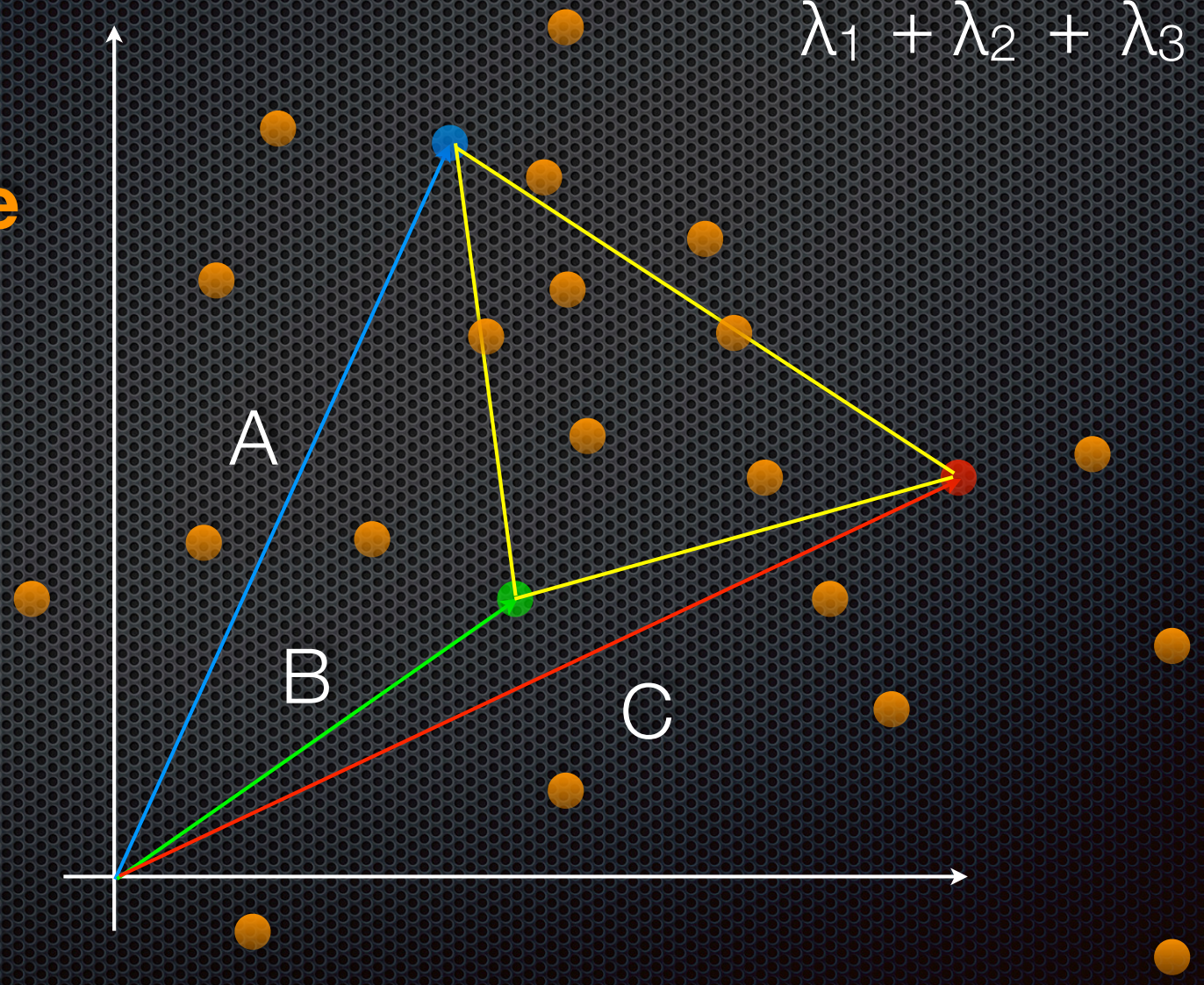
Barycentric Coordinates



Not Barycentric Coordinates

$\lambda_1 + \lambda_2 + \lambda_3 \neq 1$

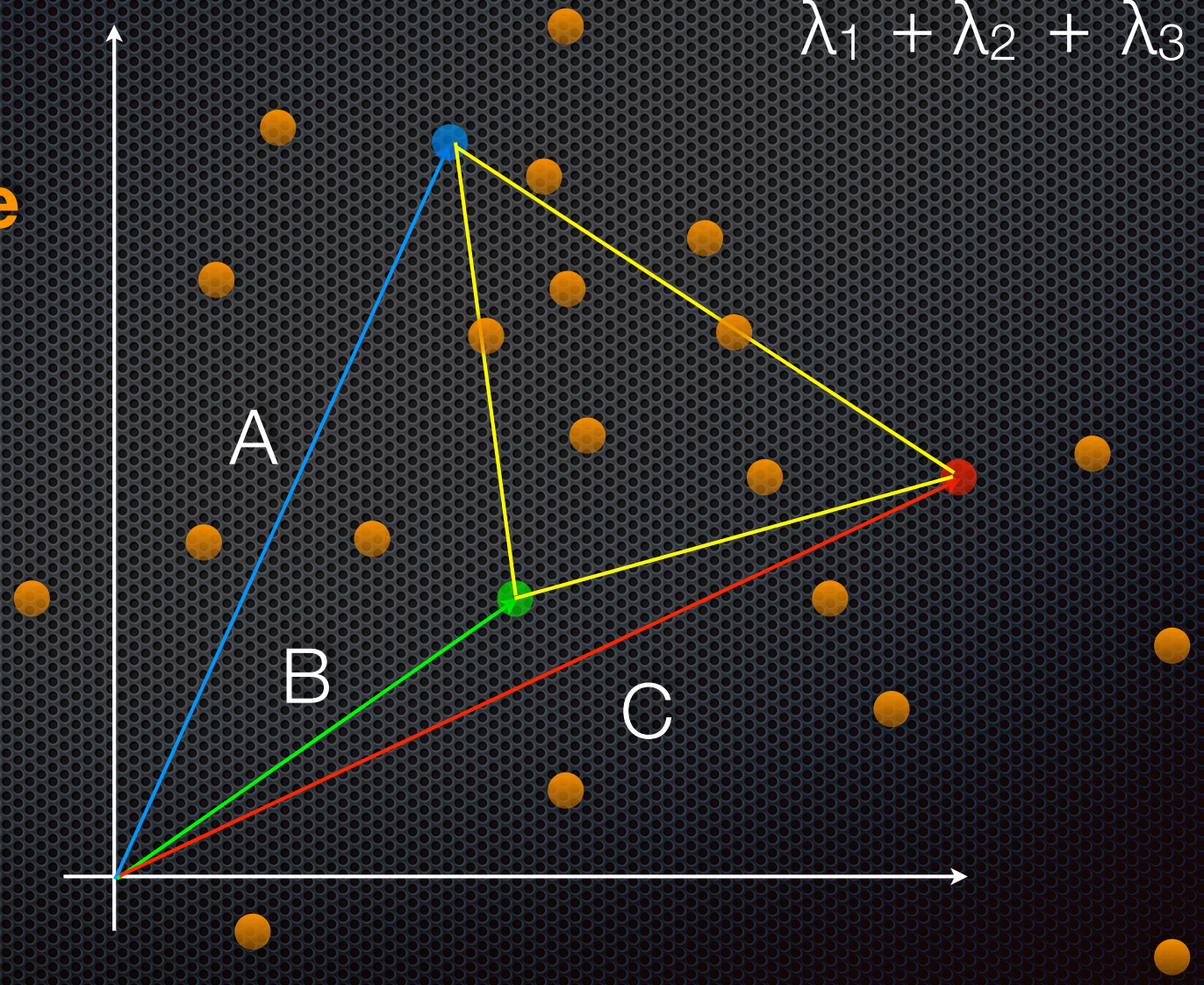
Anywhere



Linear Combination

$\lambda_1 + \lambda_2 + \lambda_3 \neq 1$

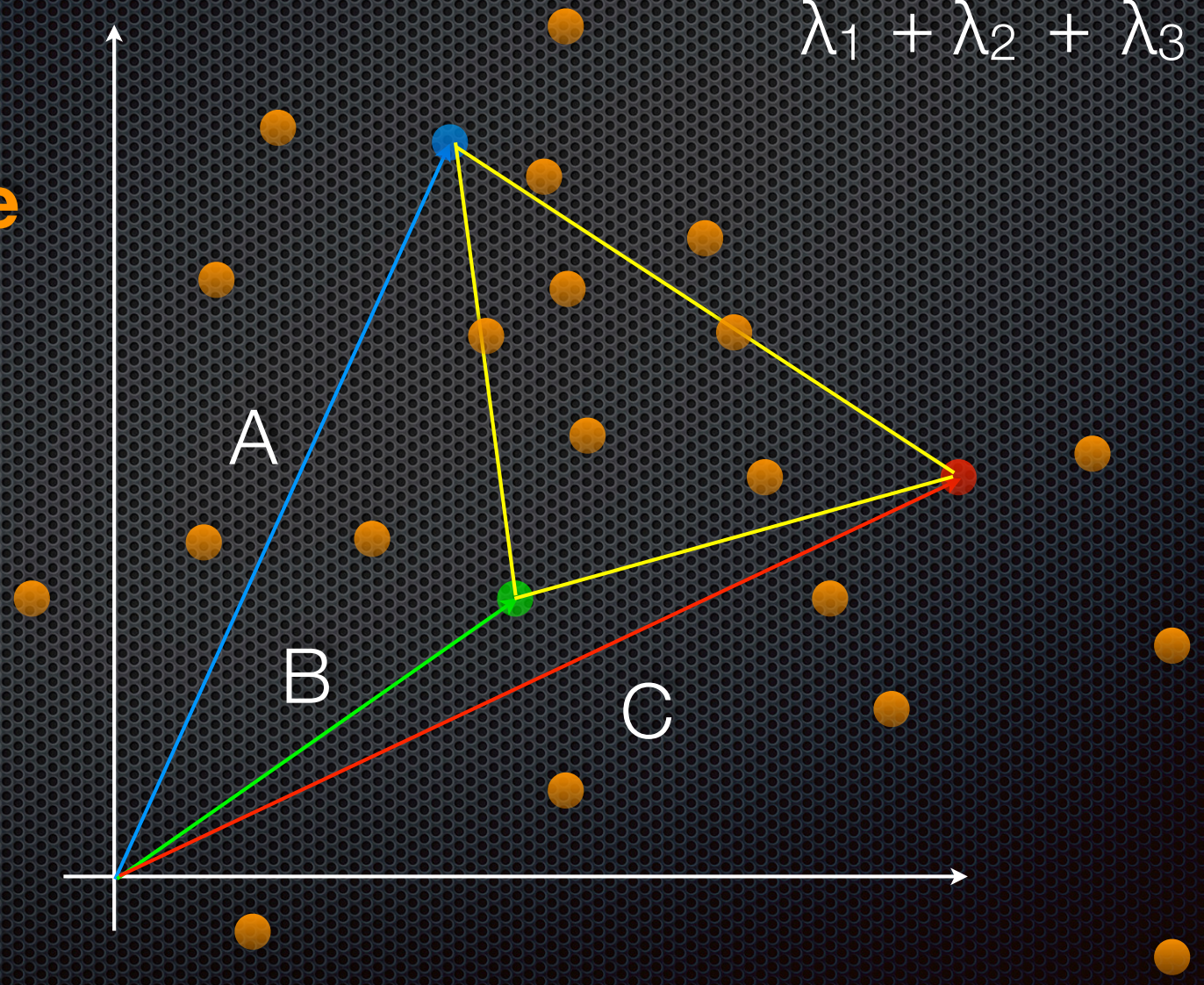
Anywhere



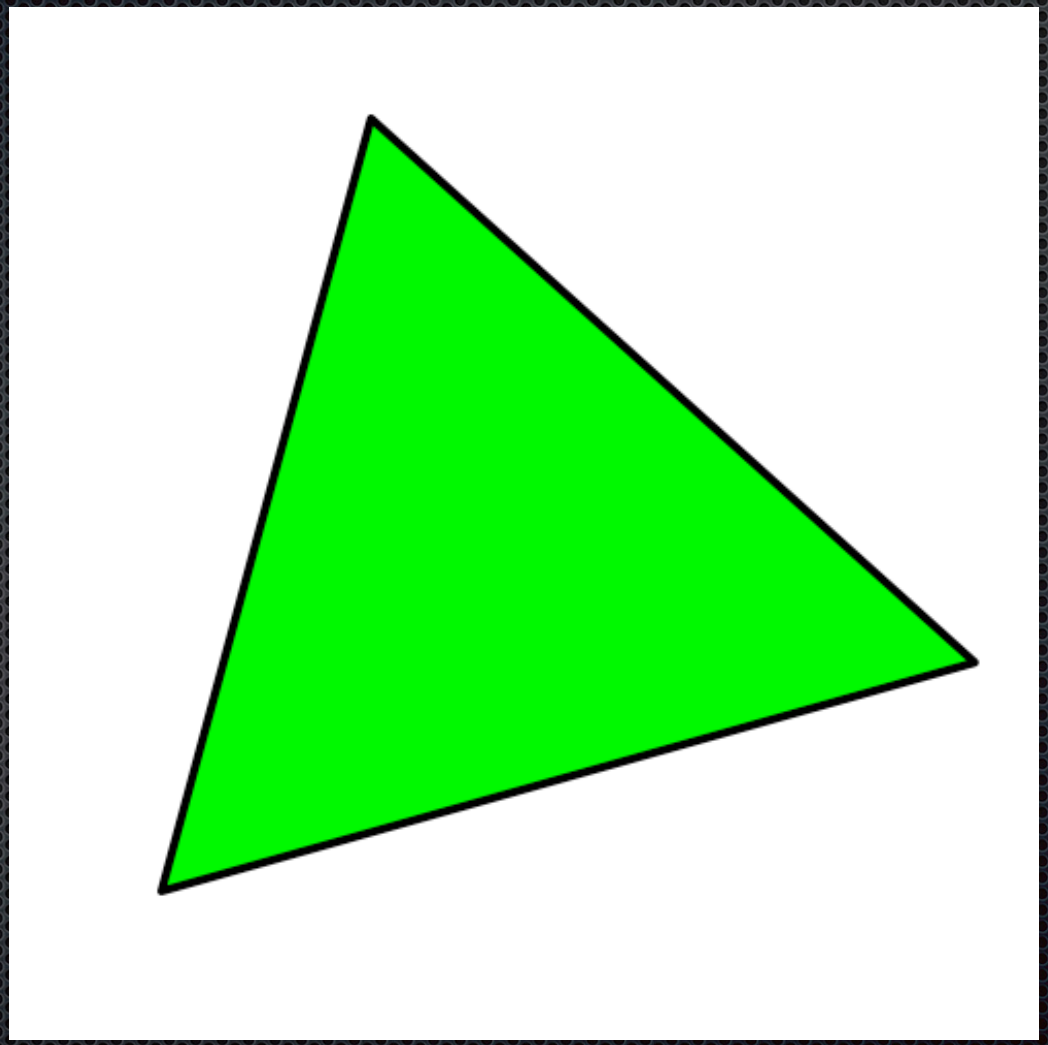
Linear Combination but **Not** a Blend

$\lambda_1 + \lambda_2 + \lambda_3 \neq 1$

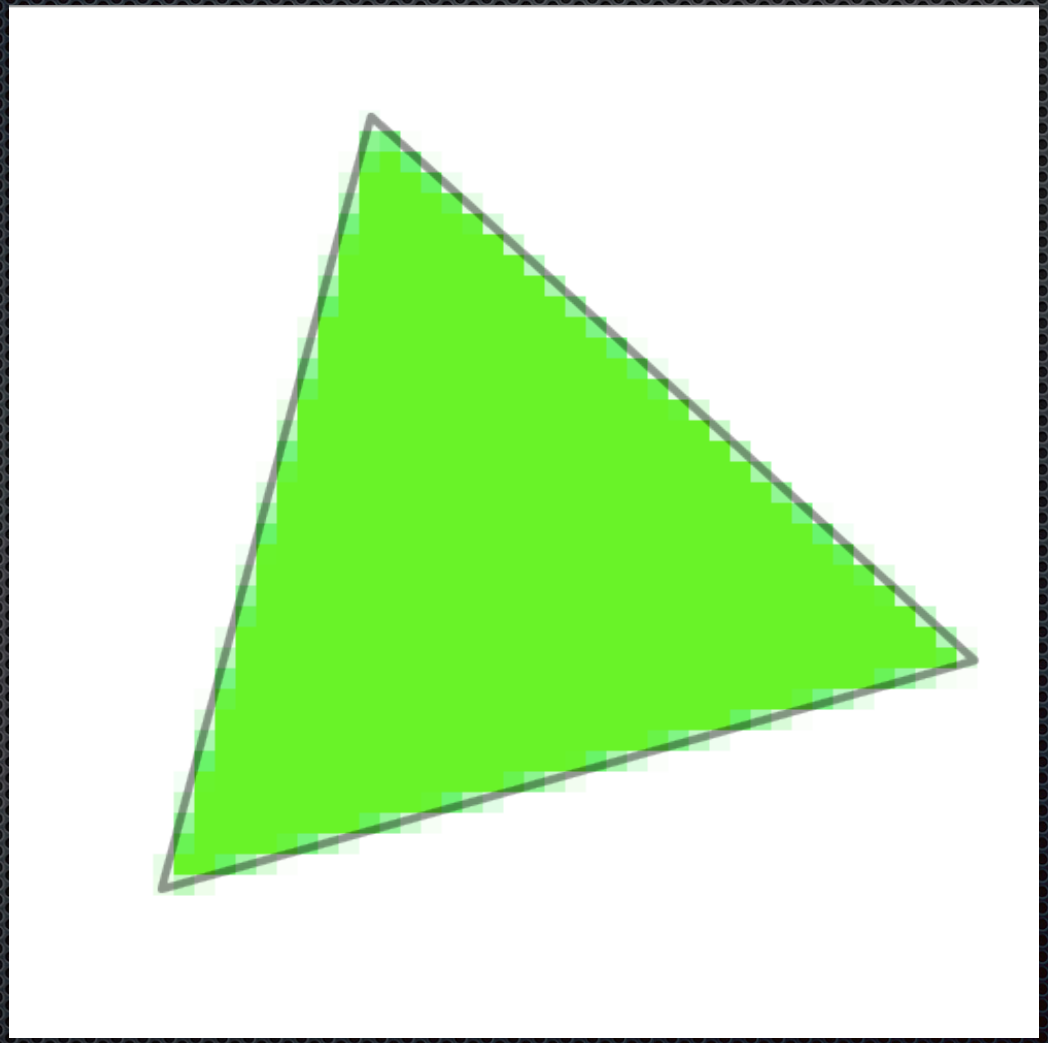
Anywhere



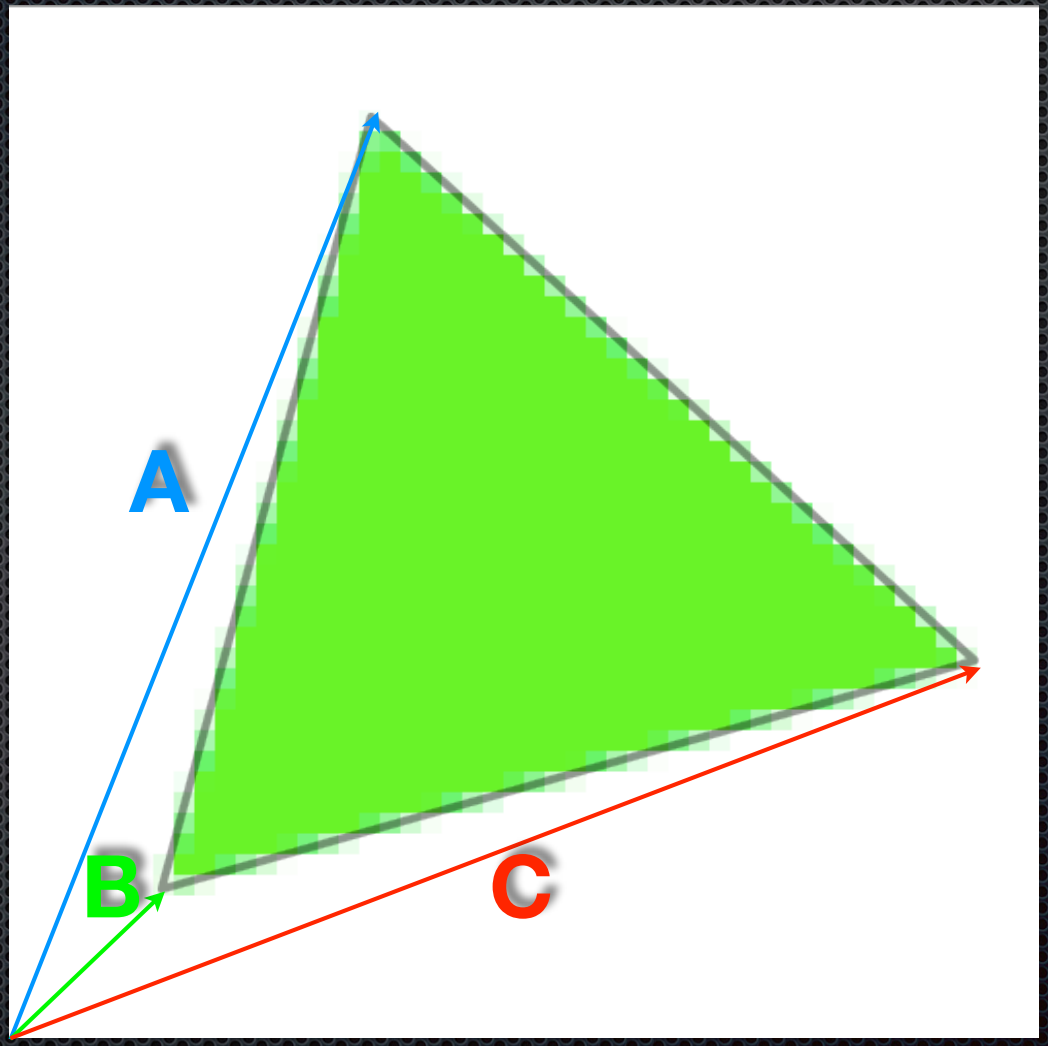
Rasterization



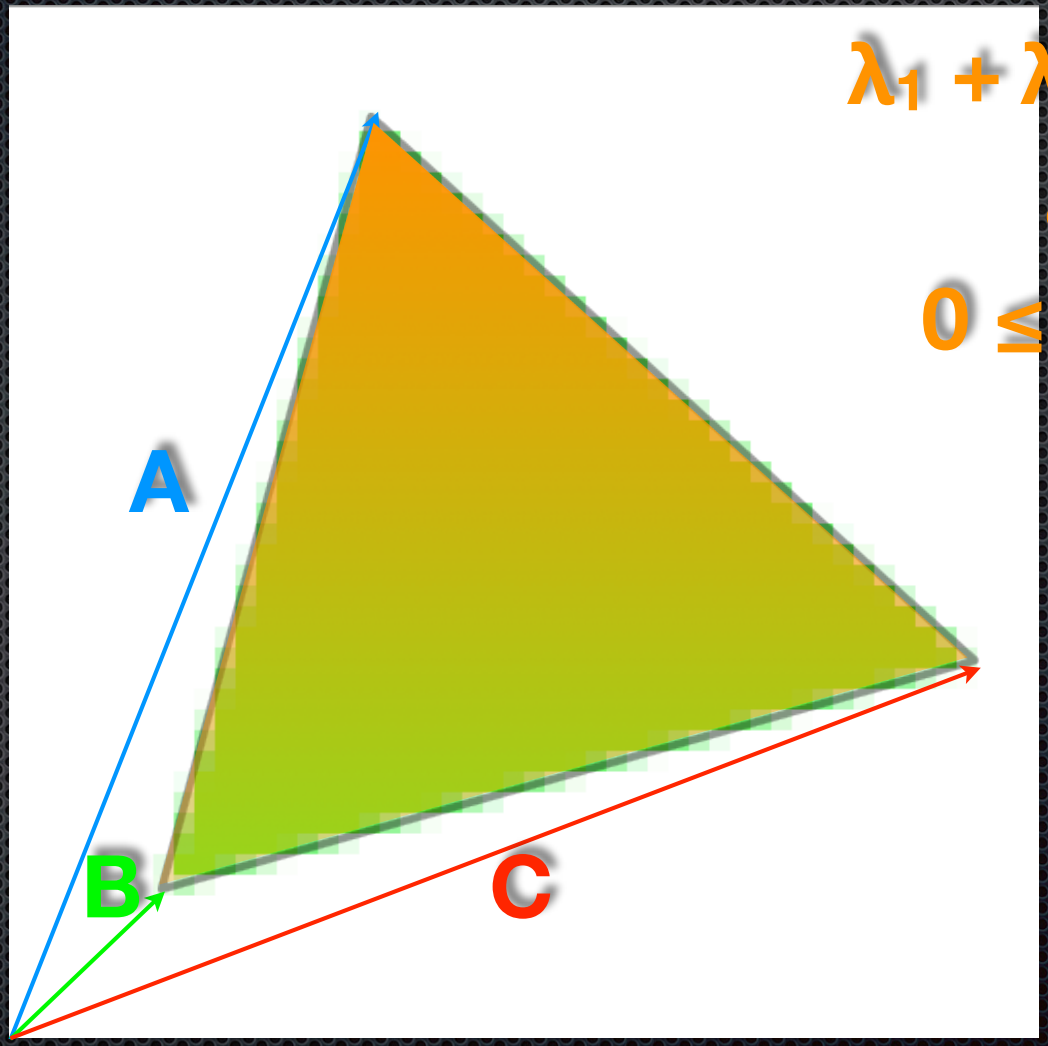
Rasterization



Rasterization



Barycentric Coordinates

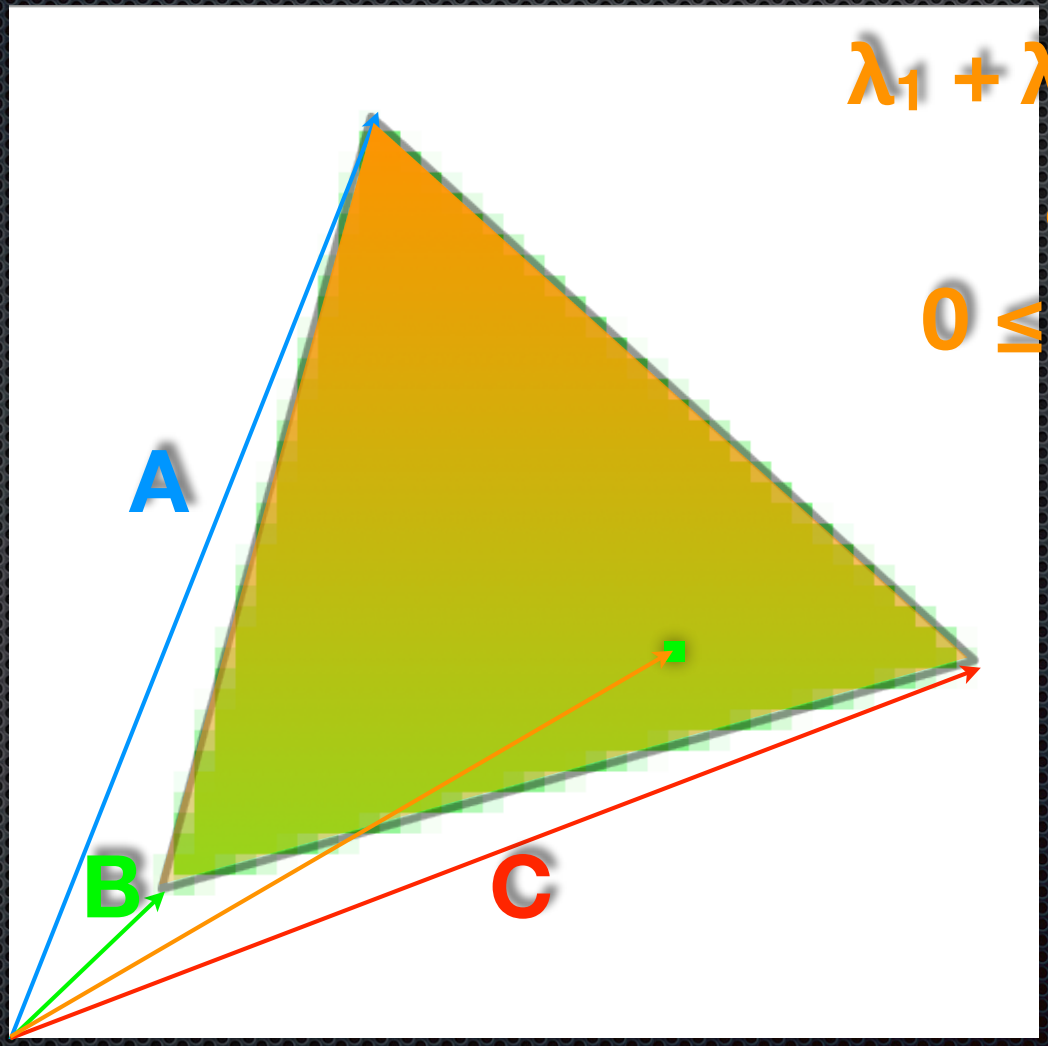


$\lambda_1 + \lambda_2 + \lambda_3 = 1$

and

$0 \leq \lambda_x \leq 1$

Barycentric Coordinates

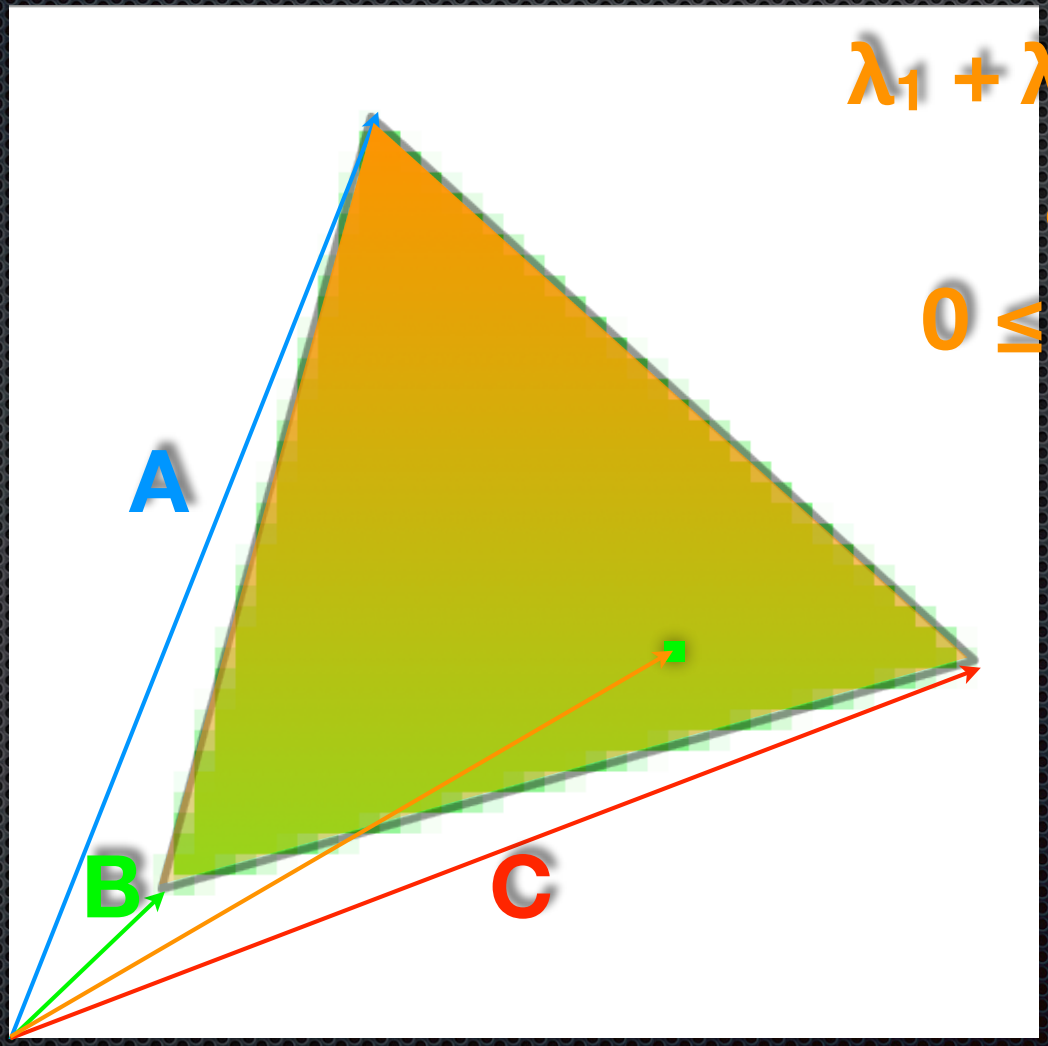


$\lambda_1 + \lambda_2 + \lambda_3 = 1$

and

$0 \leq \lambda_x \leq 1$

Barycentric Coordinates



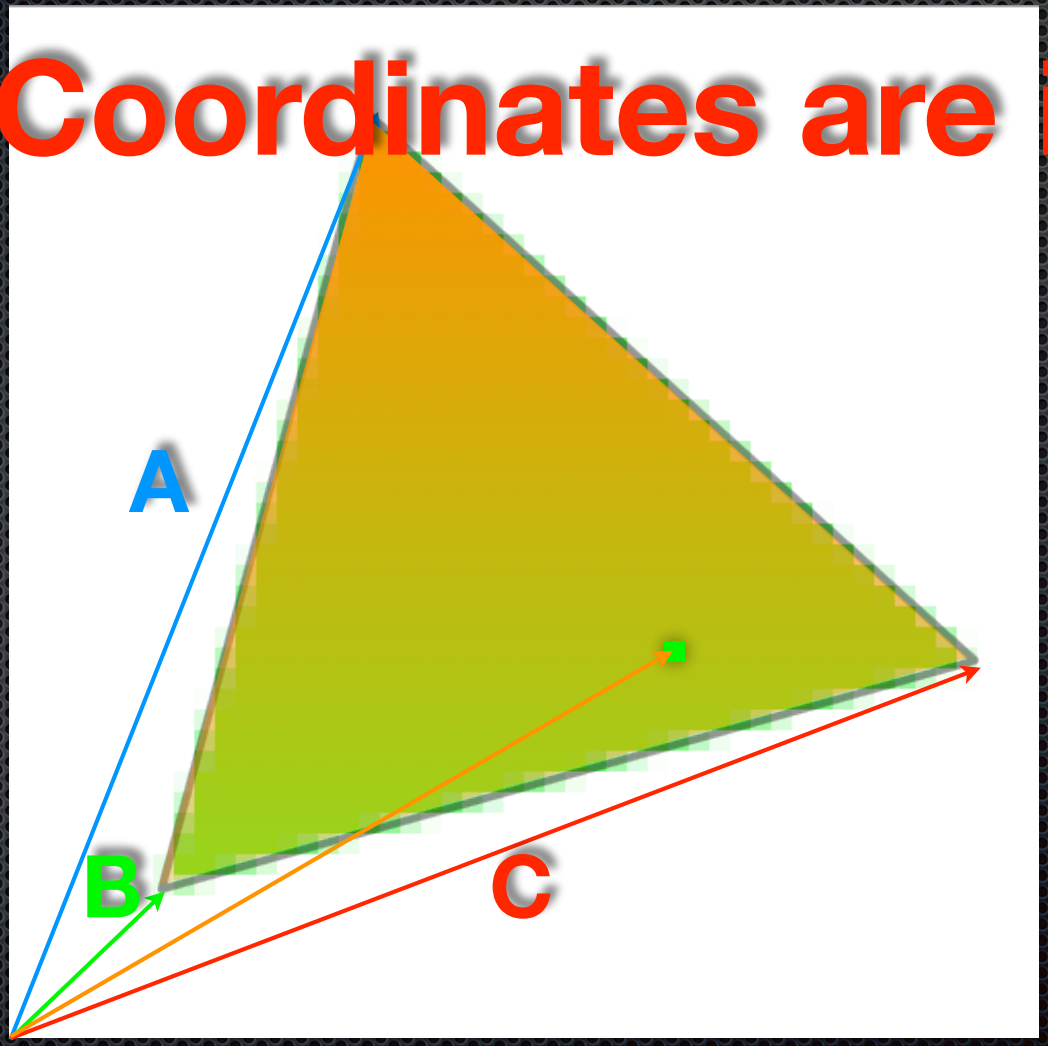
$\lambda_1 + \lambda_2 + \lambda_3 = 1$

and

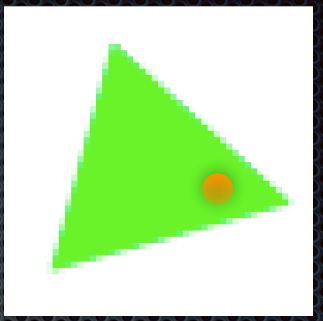
$0 \leq \lambda_x \leq 1$

Barycentric Coordinates

Pixel Coordinates are in X,Y



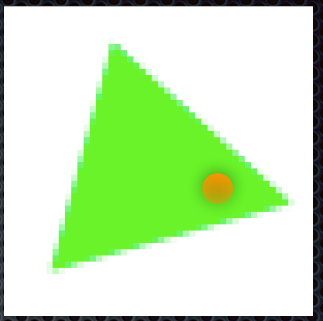
Barycentric Coordinates



$$x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3$$
$$y = \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

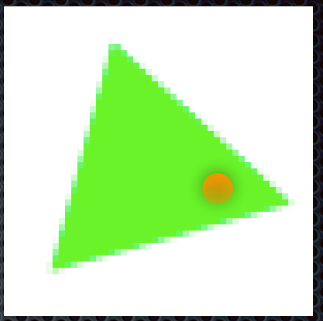
Barycentric Coordinates



$$x = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3$$
$$y = \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3$$

$$\lambda_3 = 1 - \lambda_1 - \lambda_2$$

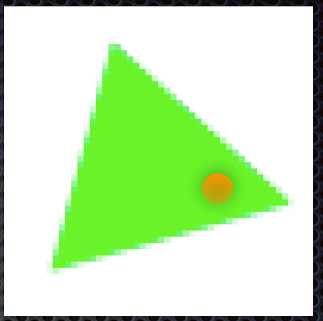
Barycentric Coordinates



$$x = \lambda_1 x_1 + \lambda_2 x_2 + (1 - \lambda_1 + \lambda_2) x_3$$
$$y = \lambda_1 y_1 + \lambda_2 y_2 + (1 - \lambda_1 + \lambda_2) y_3$$

$$\lambda_3 = 1 - \lambda_1 - \lambda_2$$

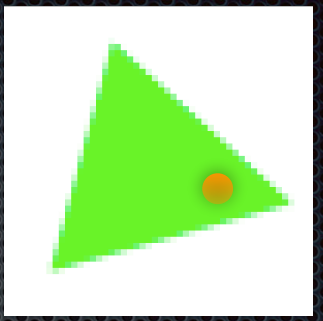
Barycentric Coordinates



Lots of rearranging...

$\lambda_3 = 1 - \lambda_1 - \lambda_2$

Barycentric Coordinates

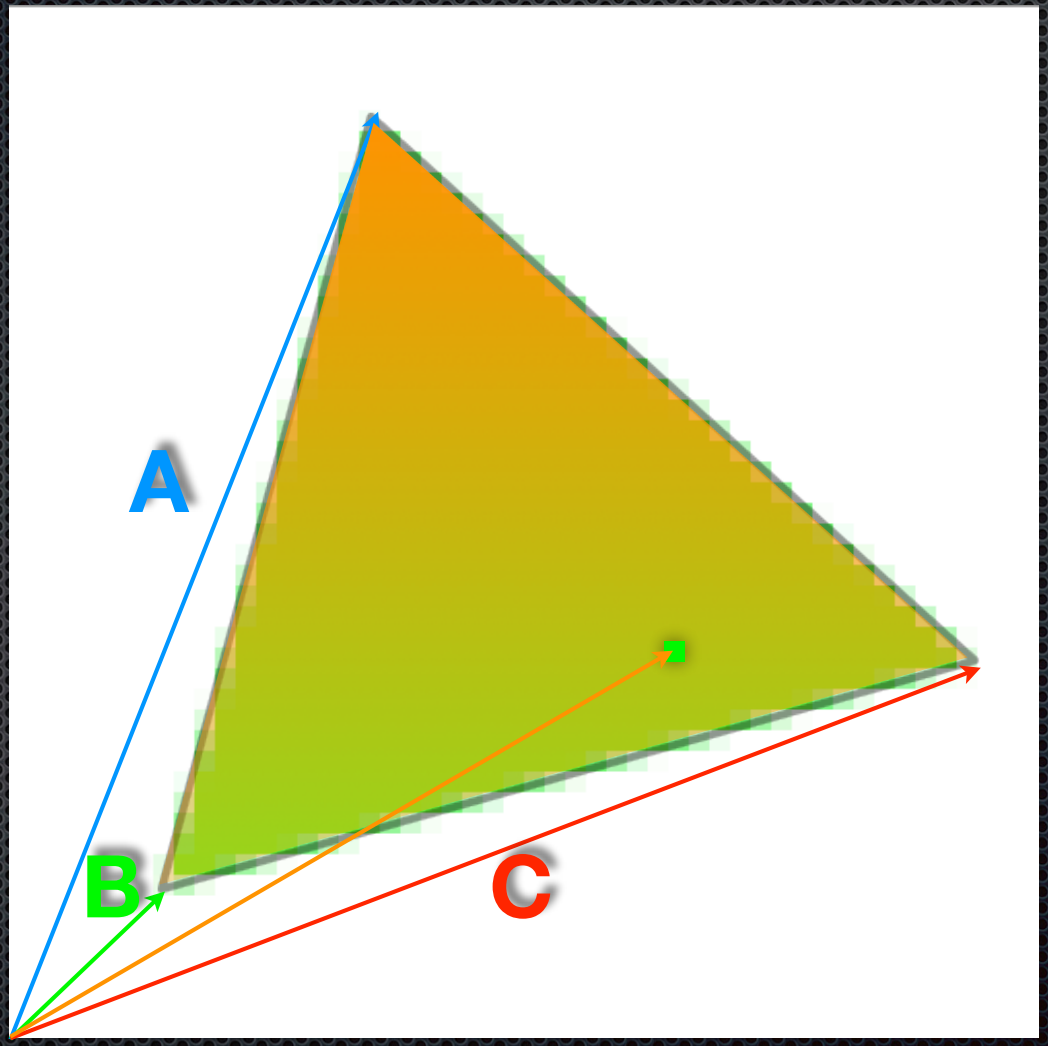


$$\lambda_1 = \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)}$$

$$\lambda_2 = \frac{(y_3 - y_1)(x - x_3) + (x_1 - x_3)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)}$$

$$\lambda_3 = 1 - \lambda_1 - \lambda_2$$

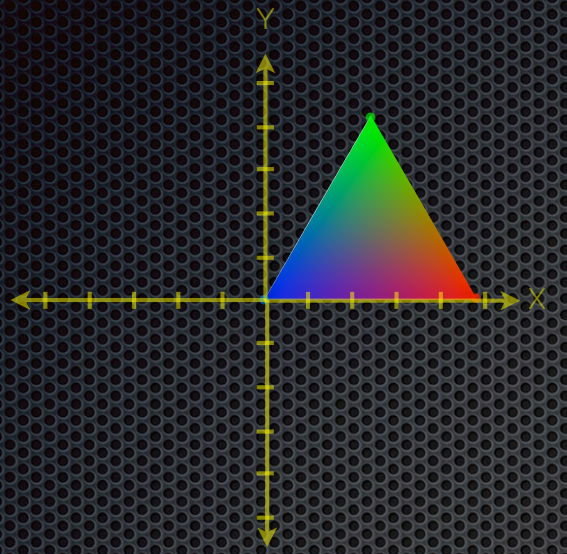
Barycentric Coordinates

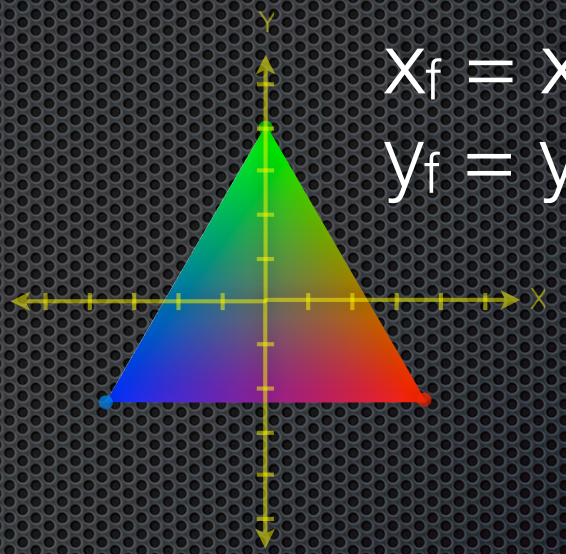


2D Transformations

Transformations

All Points



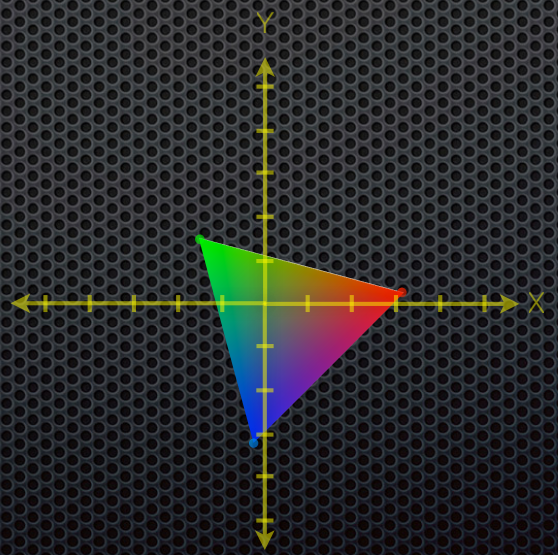


$x_f = x_o \bullet S_x$

$y_f = y_o \bullet S_y$

$x_f = x_o + t_x$

$y_f = y_o + t_y$

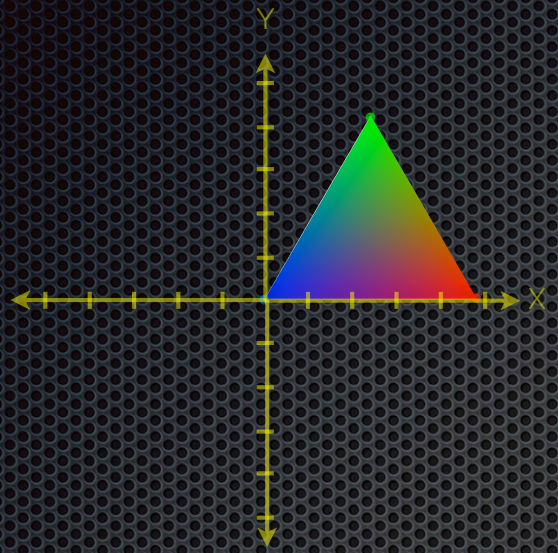


$x_f = x_i \bullet \cos\theta - y_i \bullet \sin\theta$

$y_f = x_i \bullet \sin\theta + y_i \bullet \cos\theta$

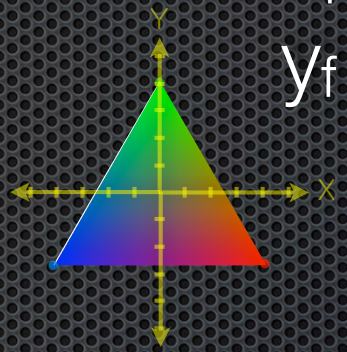
Transformations

All Points



$x_f = x_o + t_x$

$y_f = y_o + t_y$



$x_f = x_o \bullet S_x$

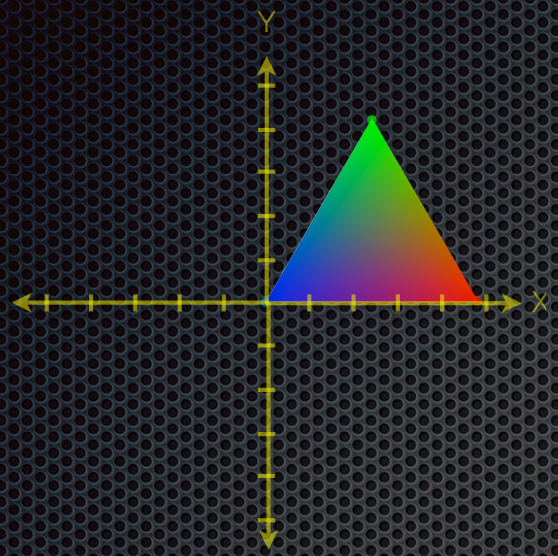
$y_f = y_o \bullet S_y$

$x_f = x_i \bullet \cos\theta - y_i \bullet \sin\theta$

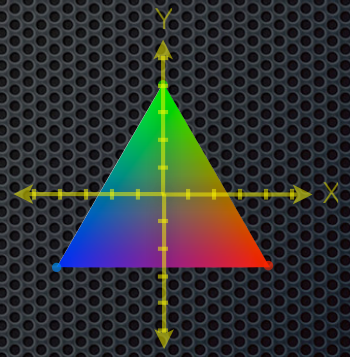
$y_f = x_i \bullet \sin\theta + y_i \bullet \cos\theta$

Transformations

All Points



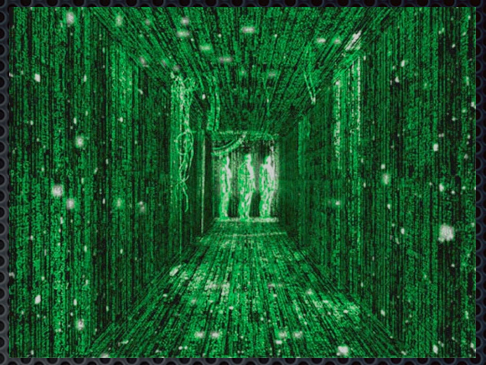
$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



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

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

Matrix Composition



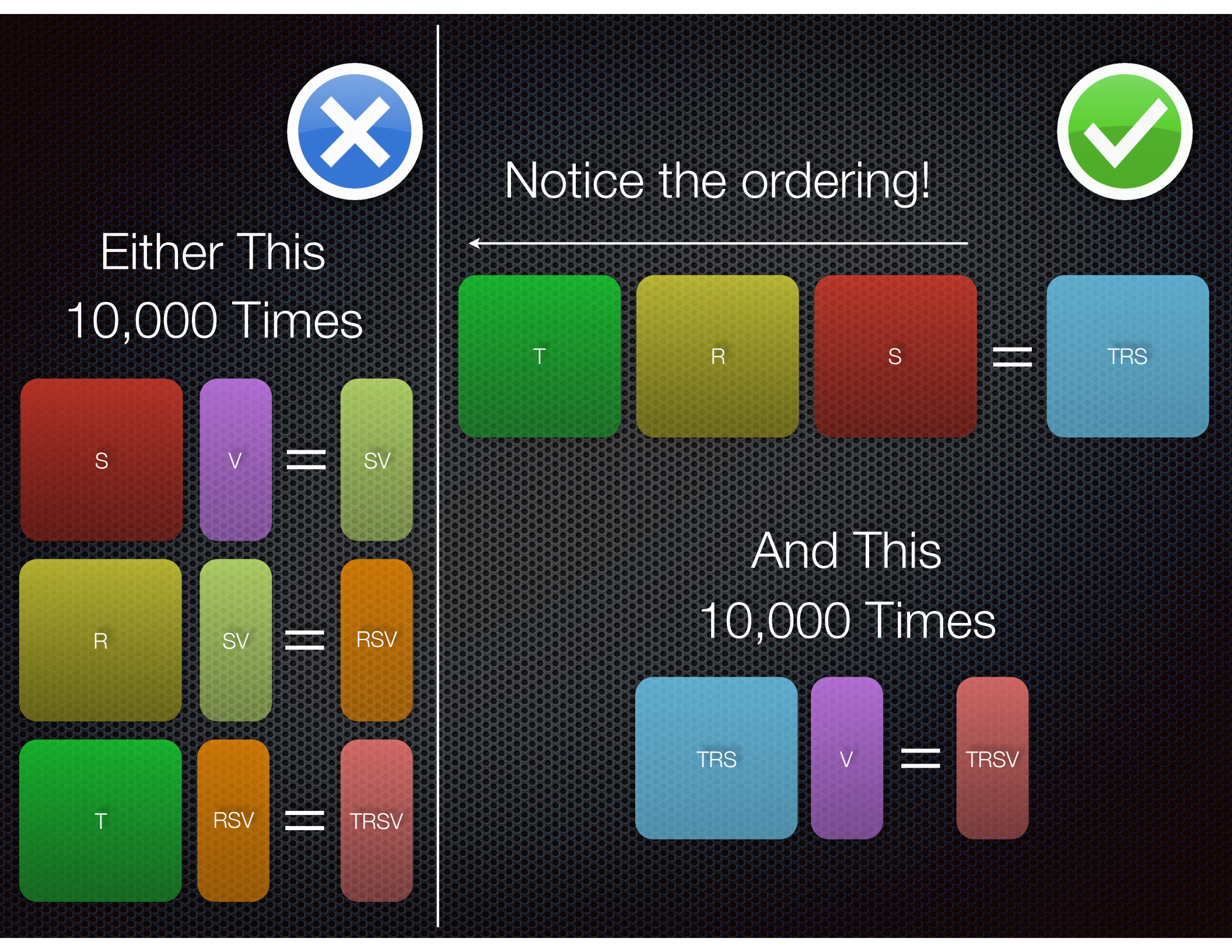
2.1 X
-3.6 Y
0.8 Z
Translate

PI/3 Rotate

2x Scale

=

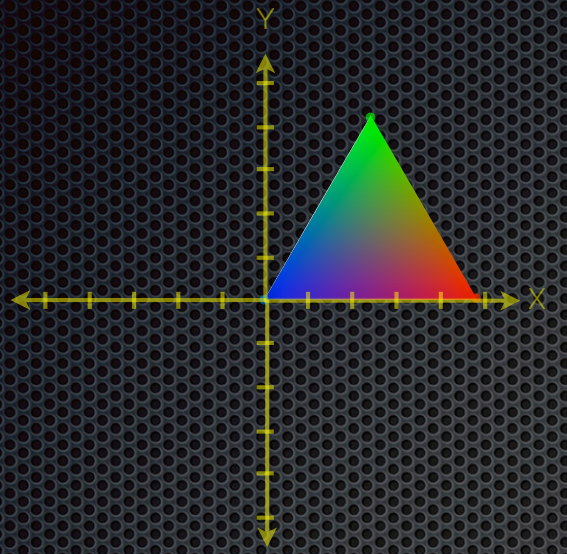
Combined Matrix That Does All 3 Ops!

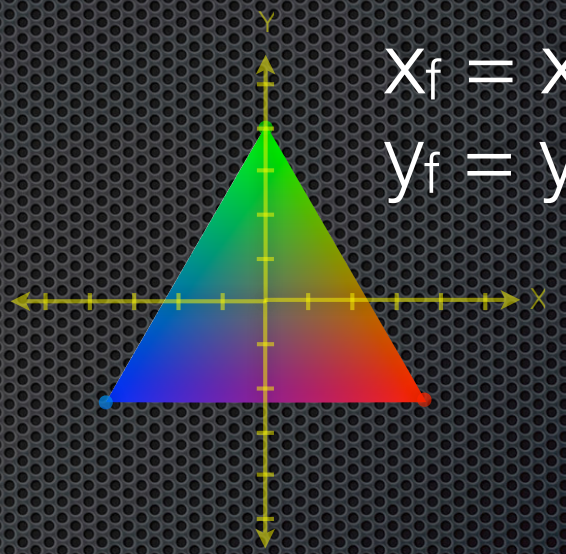

Either This
10,000 Times
S
V
SV
R
SV
RSV
T
RSV
TRSV
Notice the ordering!
T
R
S
TRS
And This
10,000 Times
TRS
V
TRSV

3D Transformations

Transformations

All Points



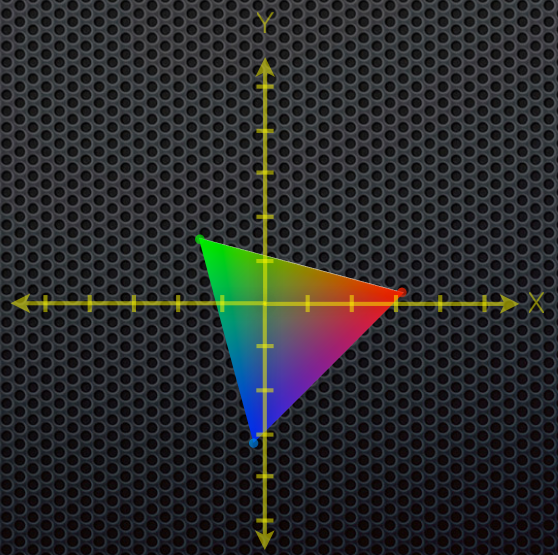


$x_f = x_o \bullet S_x$

$y_f = y_o \bullet S_y$

$x_f = x_o + t_x$

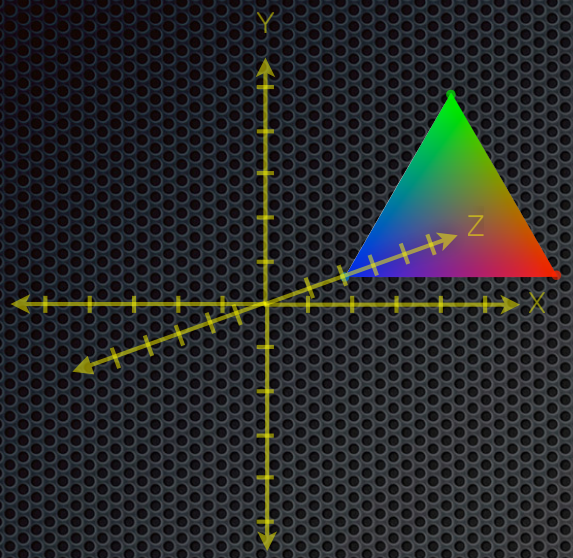
$y_f = y_o + t_y$



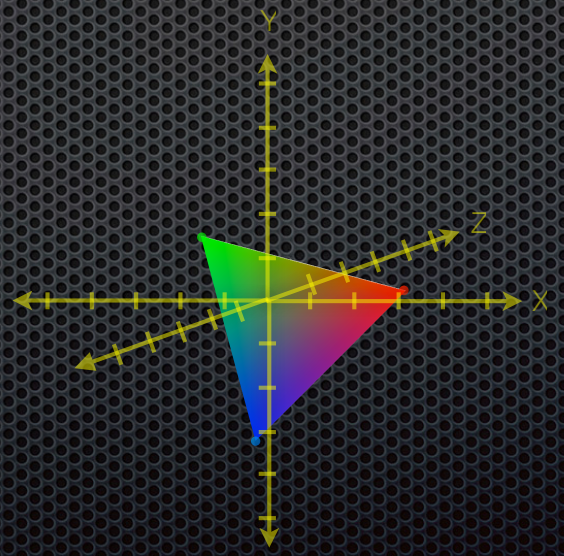
$x_f = x_i \bullet \cos\theta - y_i \bullet \sin\theta$

$y_f = x_i \bullet \sin\theta + y_i \bullet \cos\theta$

Transformations

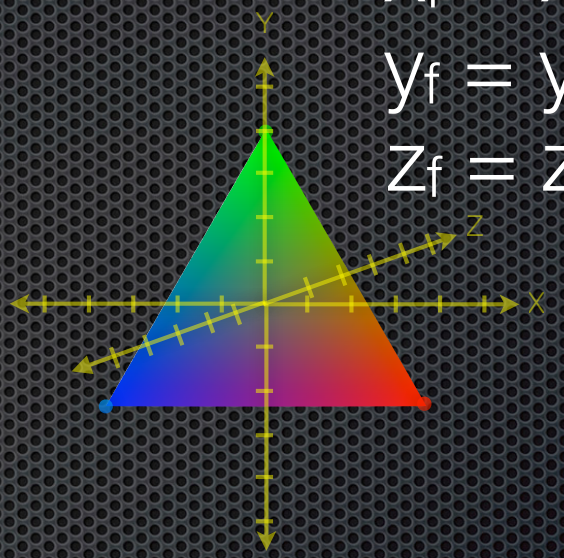


$x_f = x_o + t_x$
$y_f = y_o + t_y$
$z_f = z_o + t_z$



All Points

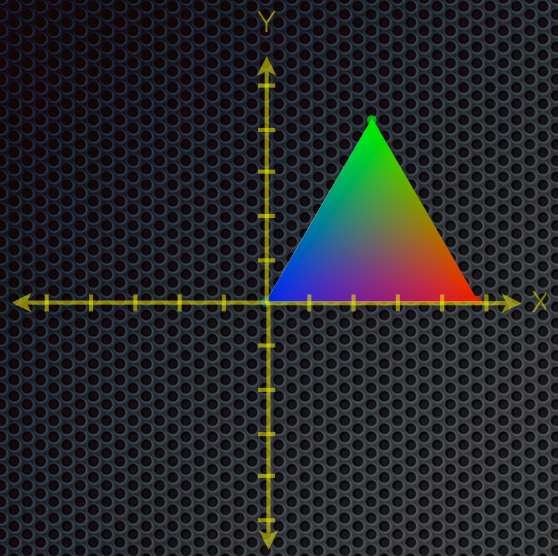
$x_f = x_o \bullet s_x$
$y_f = y_o \bullet s_y$
$z_f = z_o \bullet s_z$



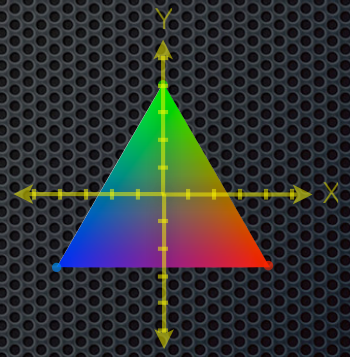
More Complex

Transformations

All Points



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

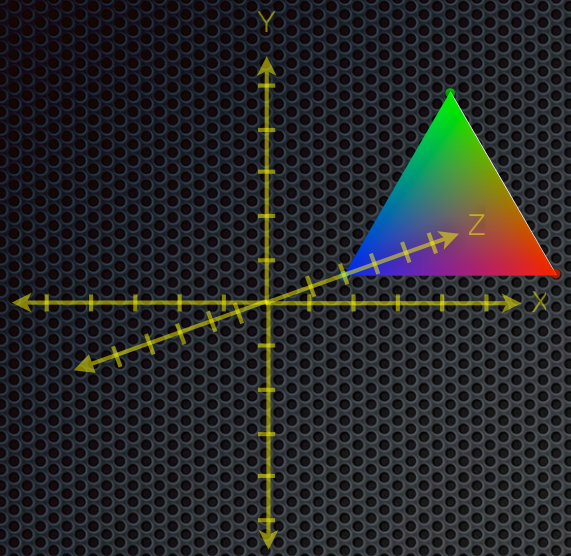


$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

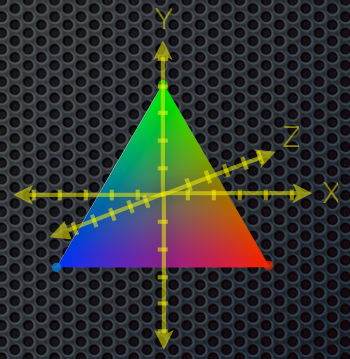
$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Transformations

All Points

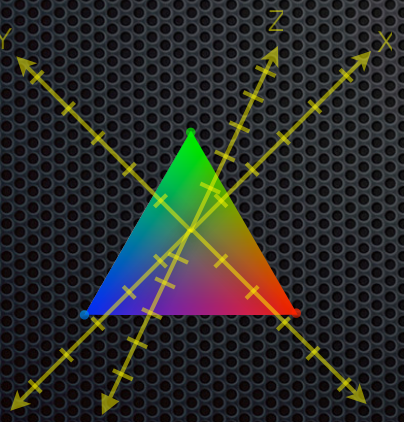


$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

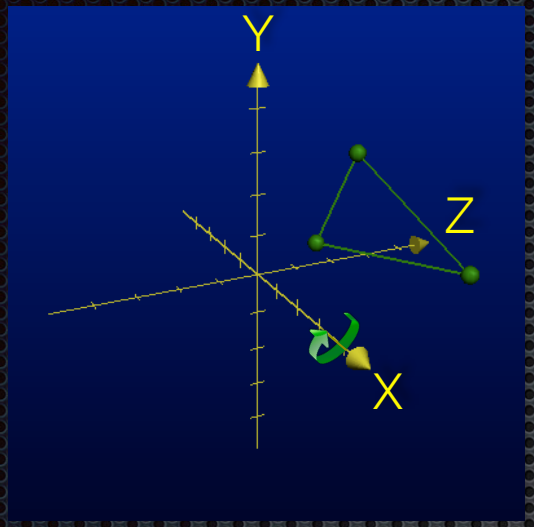
M = vector rotate matrix



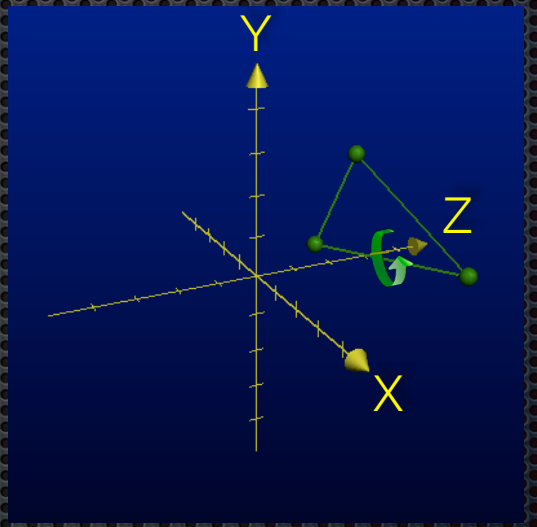
$$\begin{bmatrix} m & m & m & 0 \\ m & m & m & 0 \\ m & m & m & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation in 3D

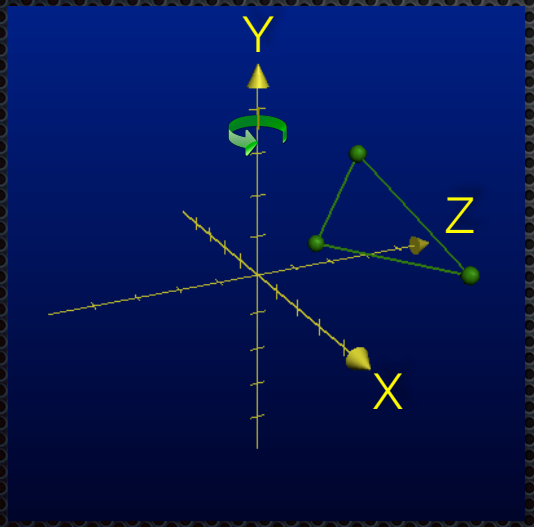
Rotation About X-Axis



Rotation About Z-Axis

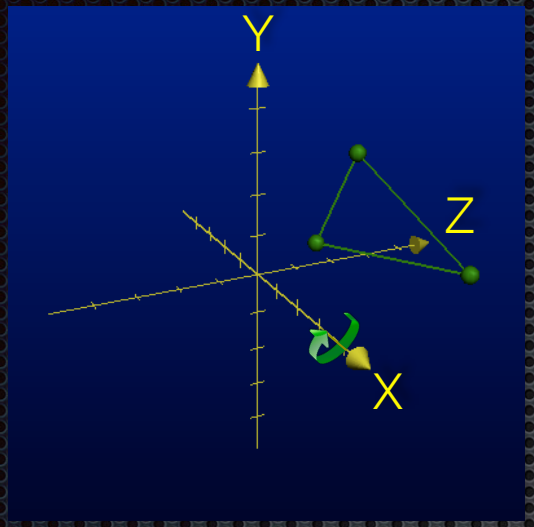


Rotation About Y-Axis

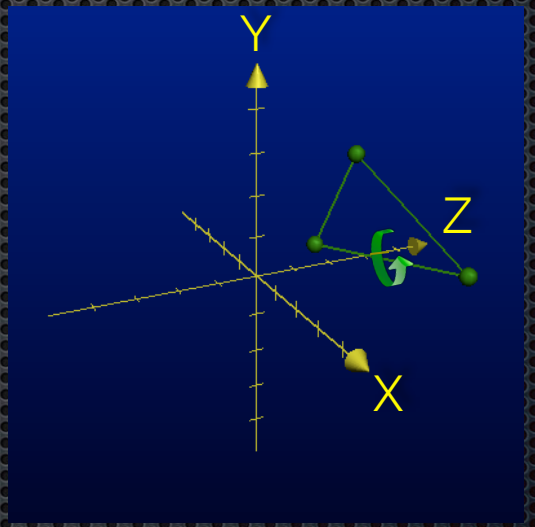


Rotation in 3D

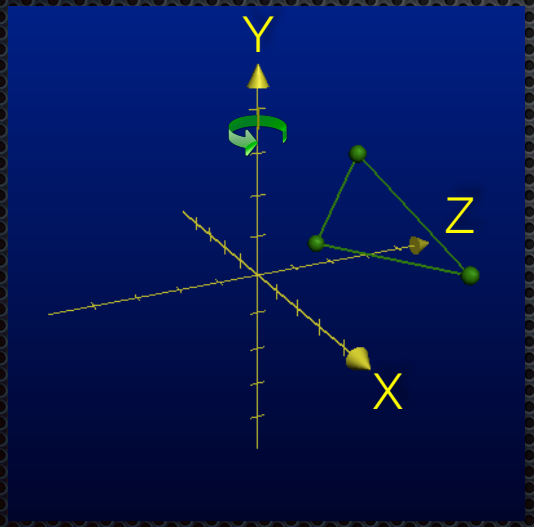
Rotation About X-Axis



Rotation About Z-Axis



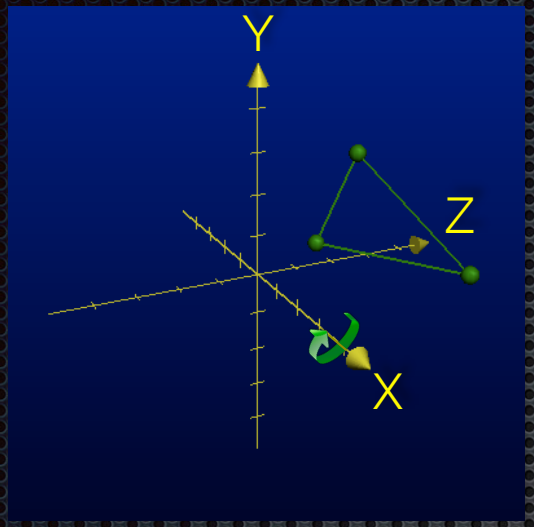
Rotation About Y-Axis



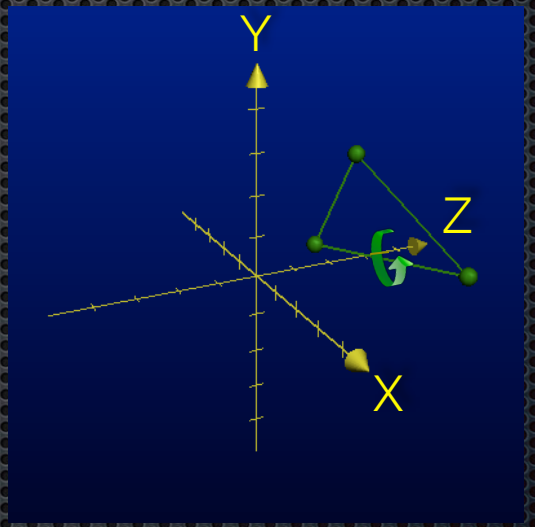
$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation in 3D

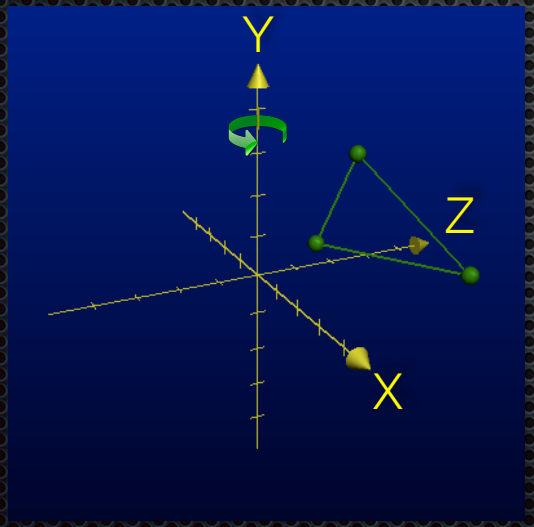
Rotation About X-Axis



Rotation About Z-Axis



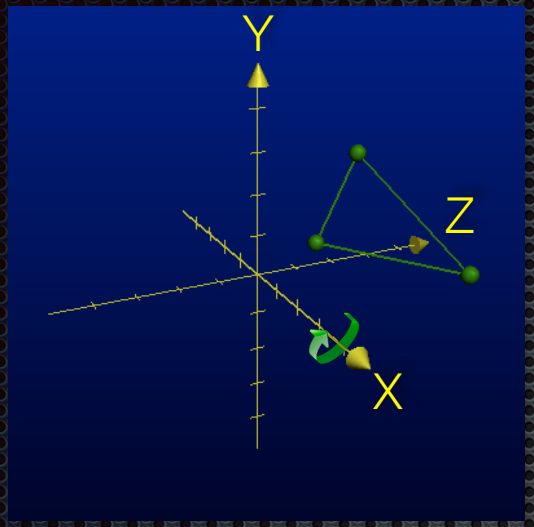
Rotation About Y-Axis



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

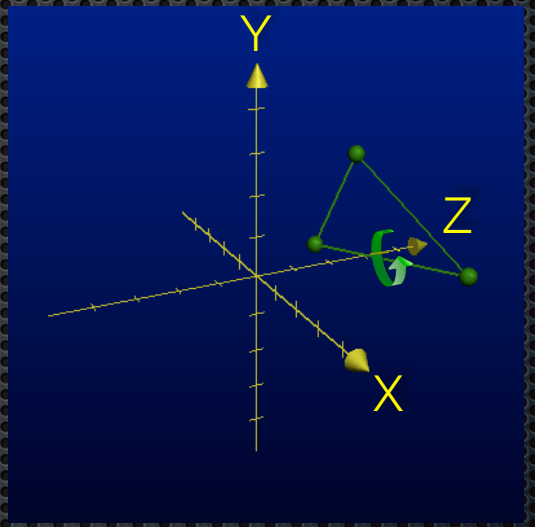
Rotation in 3D

Rotation About X-Axis

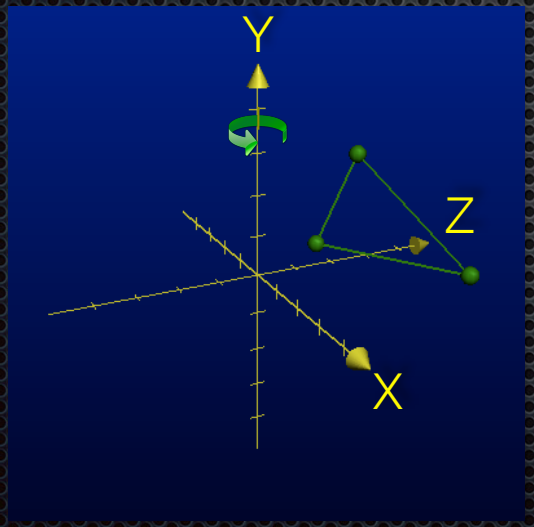


$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation About Z-Axis



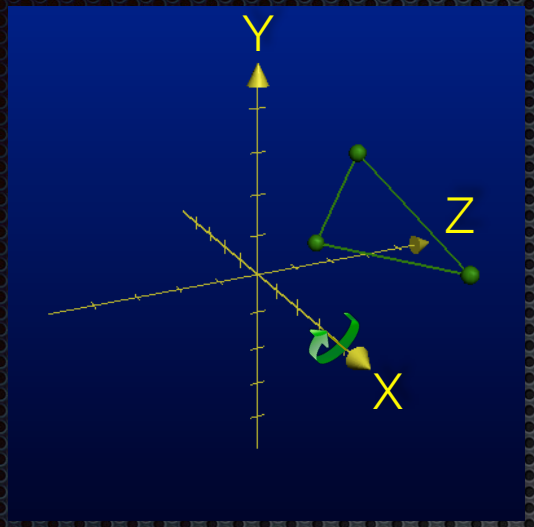
Rotation About Y-Axis



$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

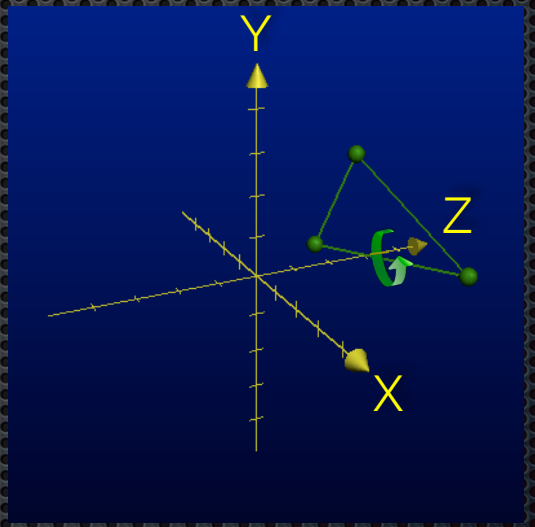
Rotation in 3D

Rotation About X-Axis

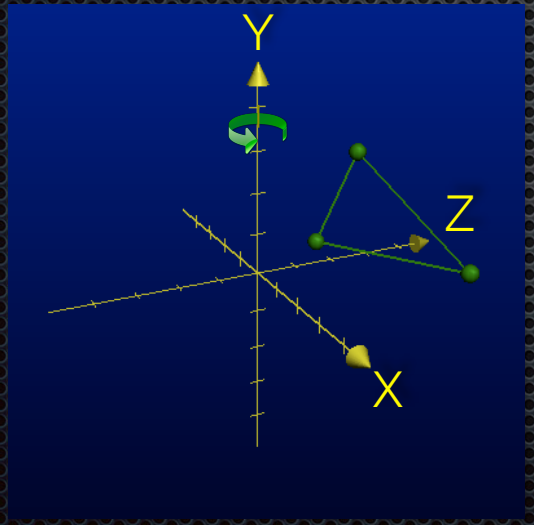


$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation About Z-Axis



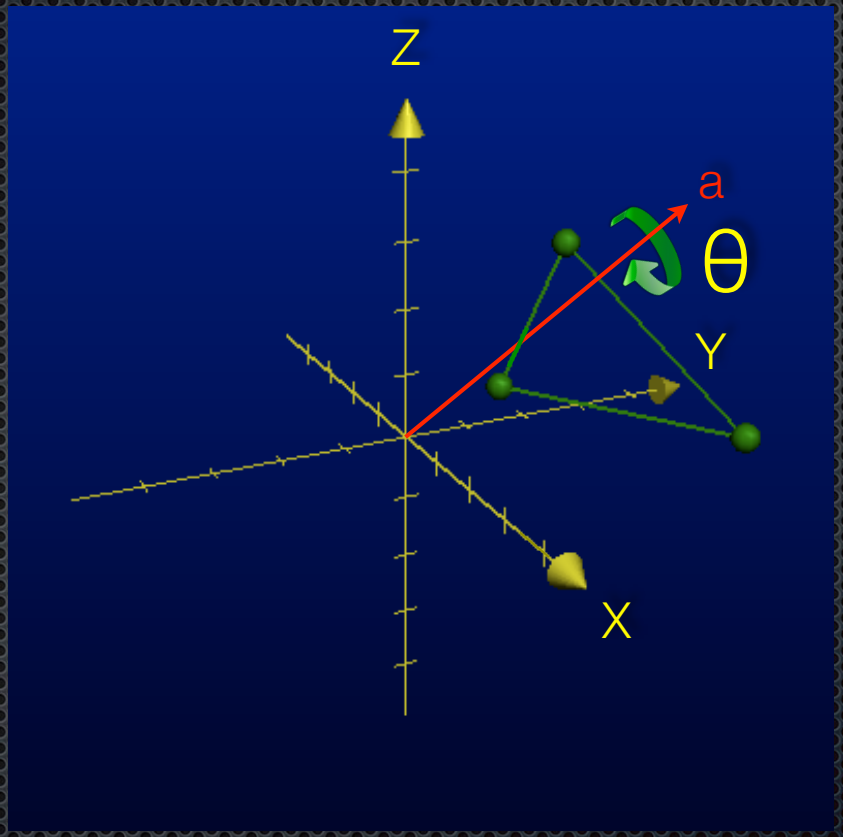
Rotation About Y-Axis



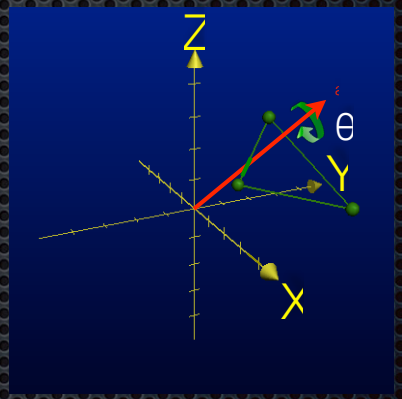
$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

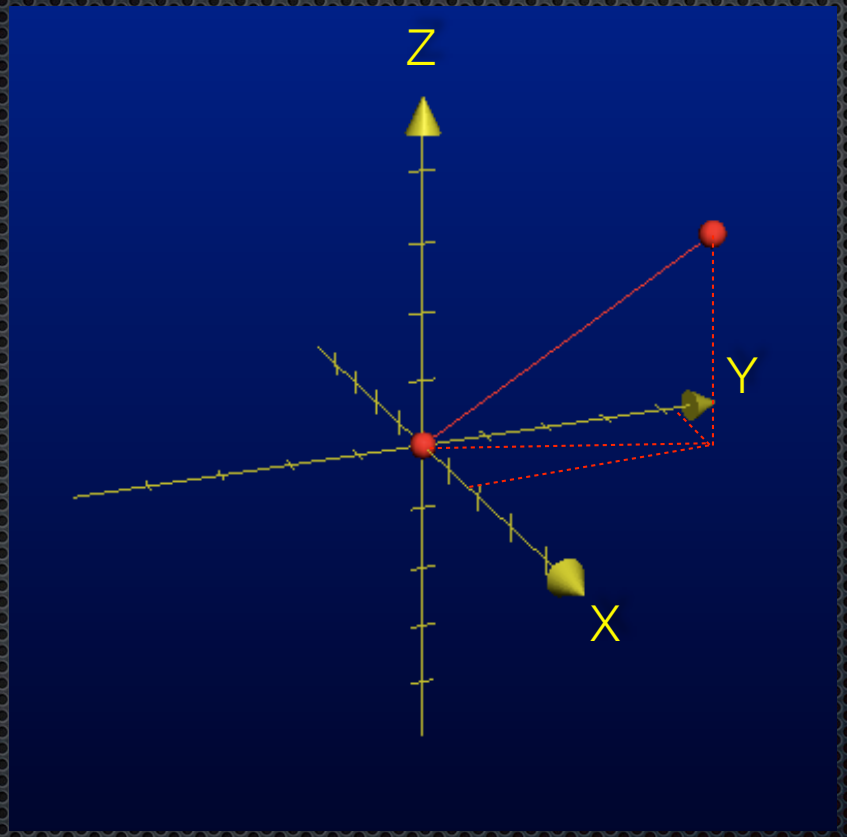
$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Arbitrary Axis Rotation in 3D

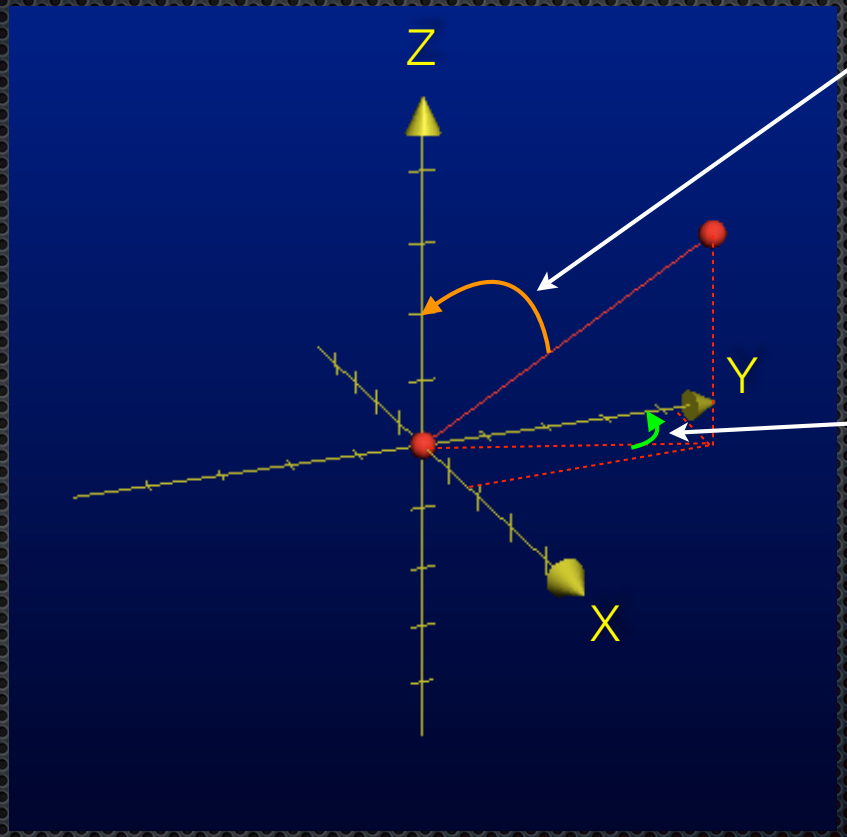


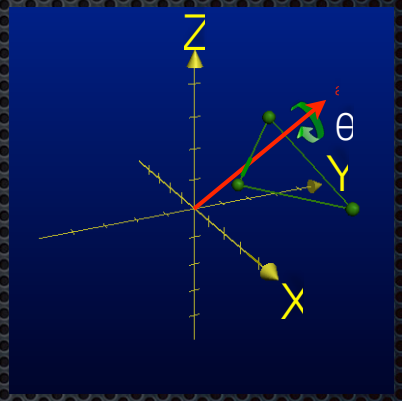
Arbitrary Axis Rotation in 3D



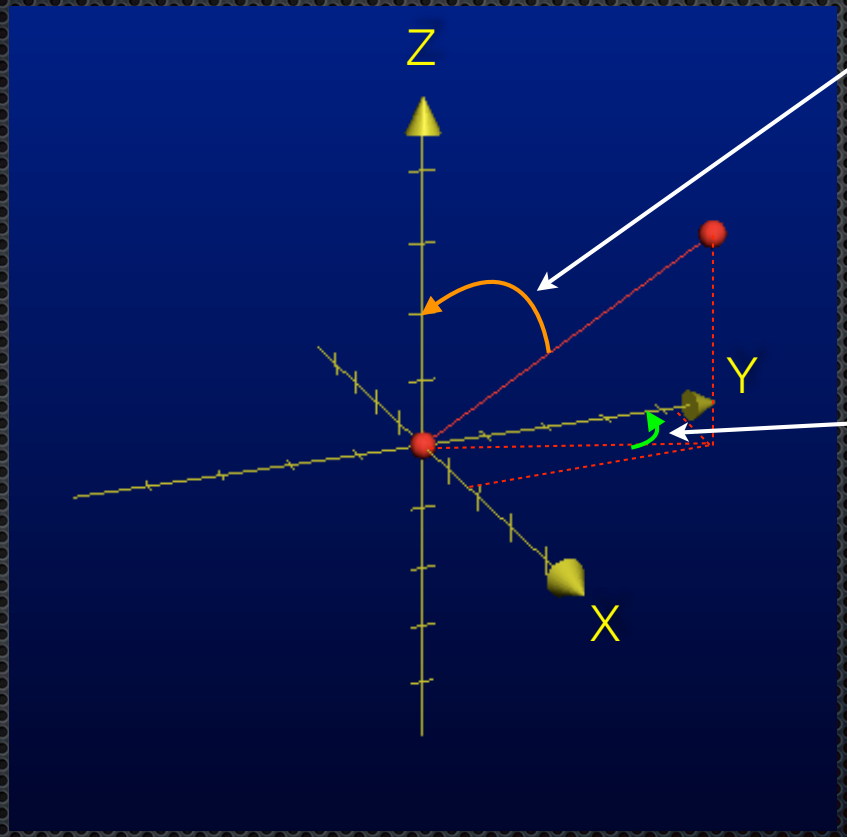


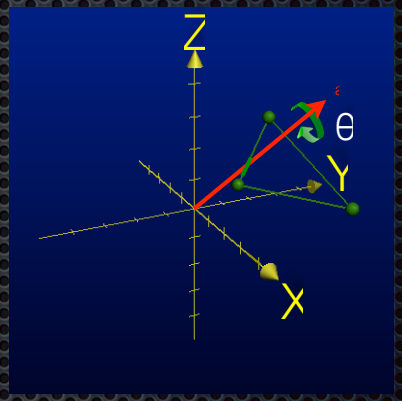
Arbitrary Axis Rotation in 3D





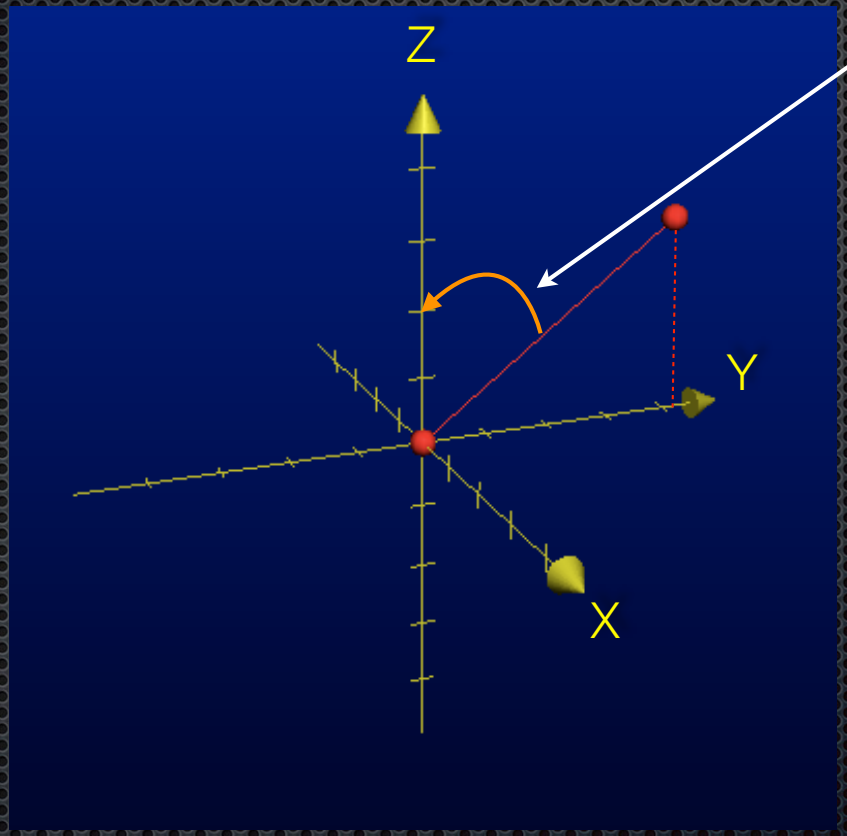
Arbitrary Axis Rotation in 3D

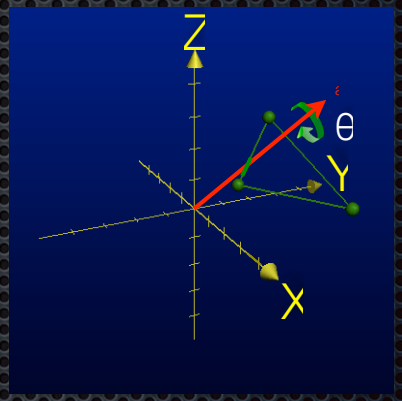




Rotate by -α

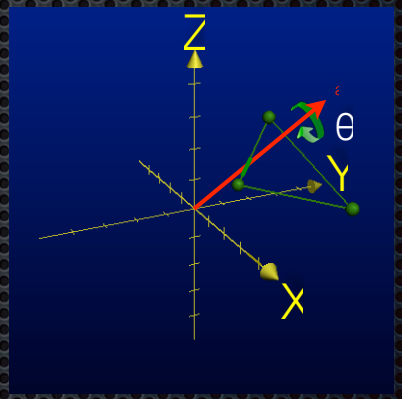
Arbitrary Axis Rotation in 3D

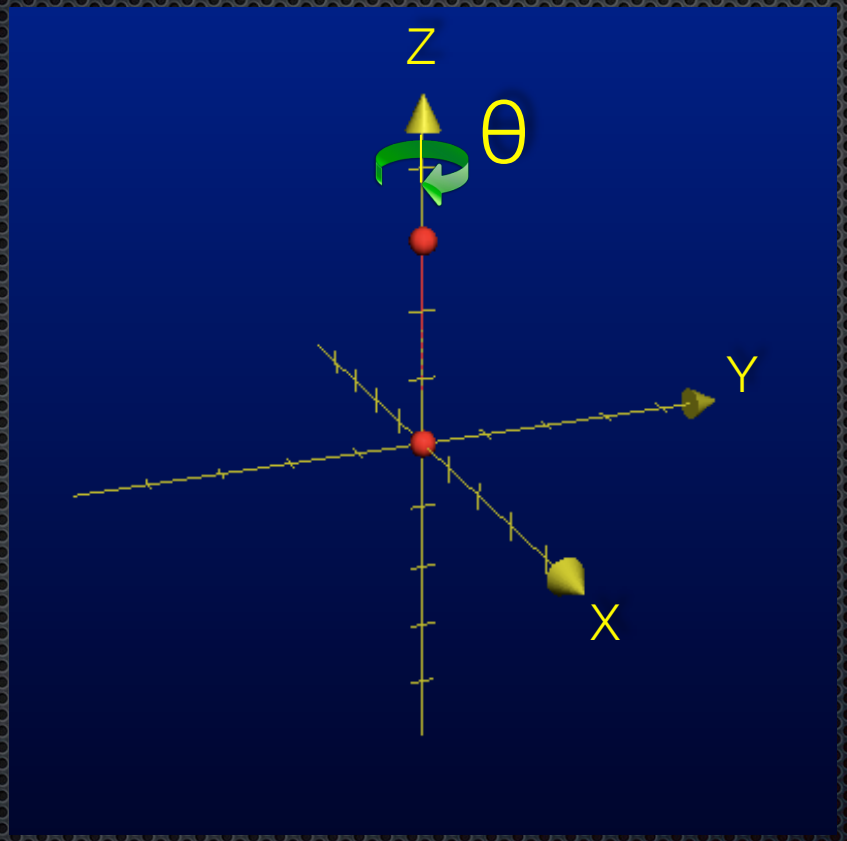




Rotate by -β

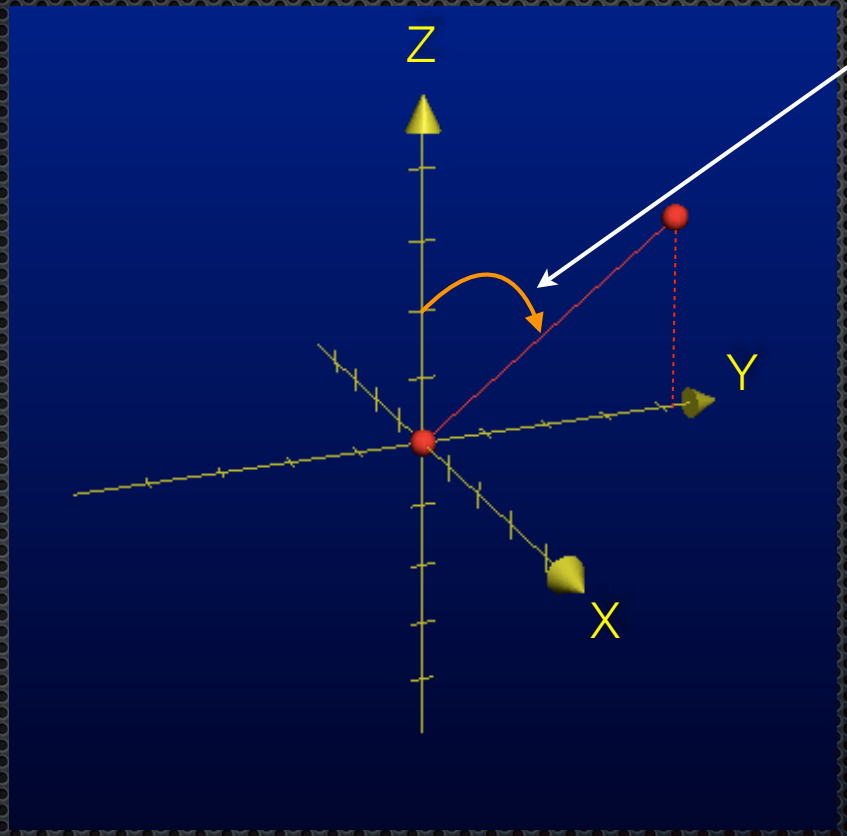
Arbitrary Axis Rotation in 3D

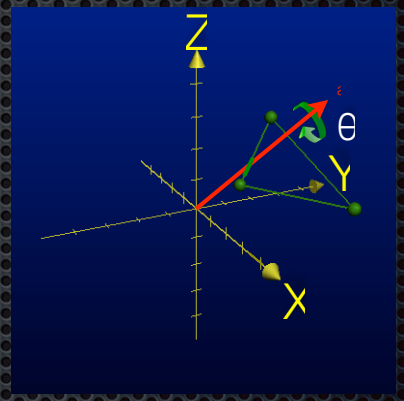




Rotate by θ

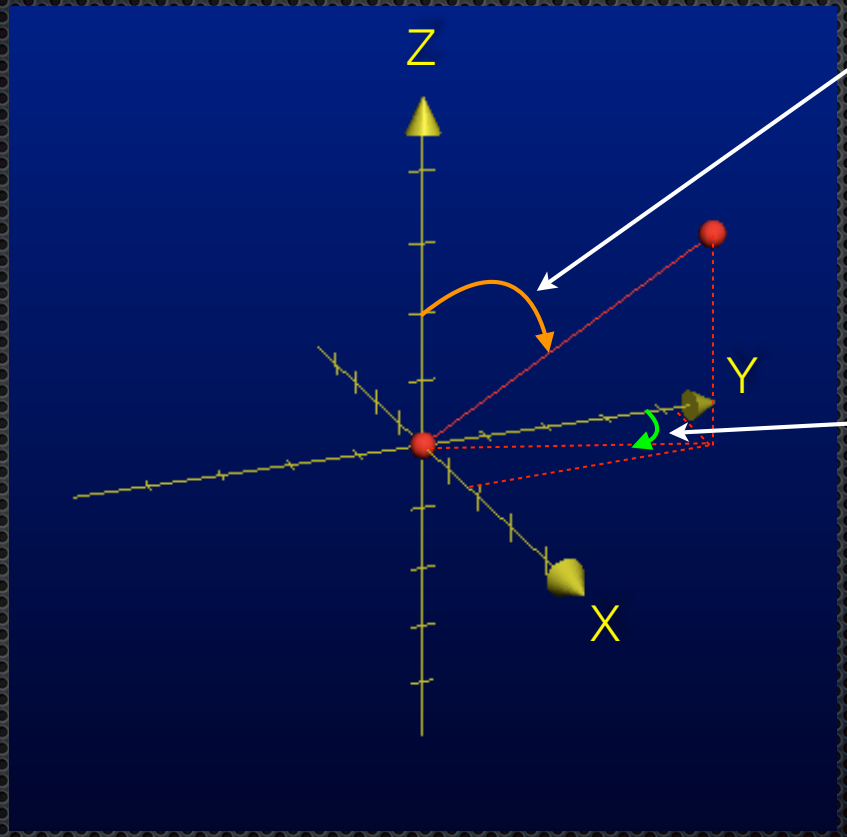
Arbitrary Axis Rotation in 3D

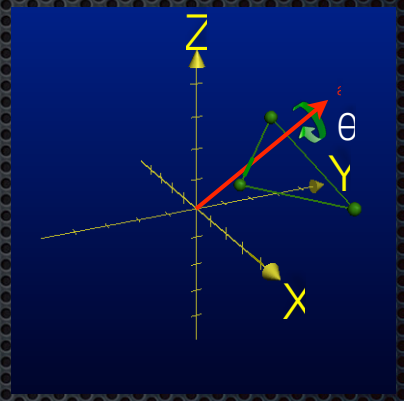




Rotate by +β

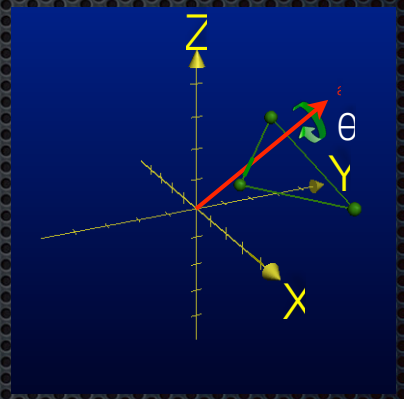
Arbitrary Axis Rotation in 3D





Rotate by +α

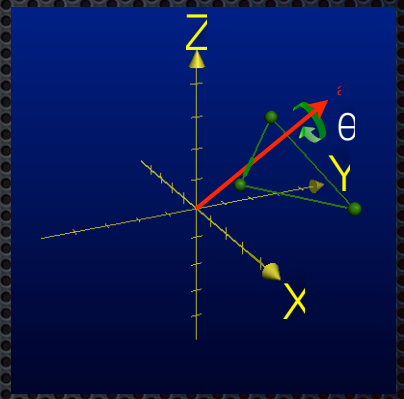
Arbitrary Axis Rotation in 3D



$$M = R_z(-\alpha) \times R_x(-\beta) \times R_z(\theta) \times R_x(\beta) \times R_z(\alpha)$$

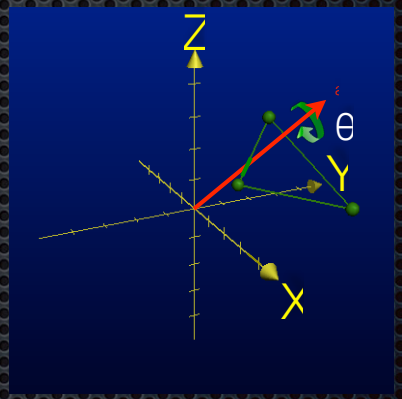
Arbitrary Axis Rotation in 3D

$$M = \begin{bmatrix} \cos-\alpha & -\sin-\alpha & 0 & 0 \\ \sin-\alpha & \cos-\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos-\beta & -\sin-\beta & 0 \\ 0 & \sin-\beta & \cos-\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



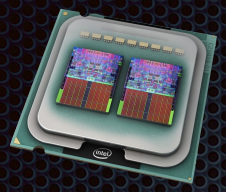
$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta & 0 \\ 0 & \sin\beta & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Arbitrary Axis Rotation in 3D

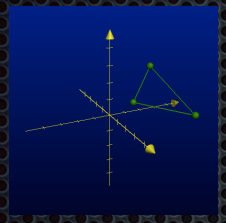


$$M = \begin{bmatrix} m & m & m & 0 \\ m & m & m & 0 \\ m & m & m & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Model View Matrix Program



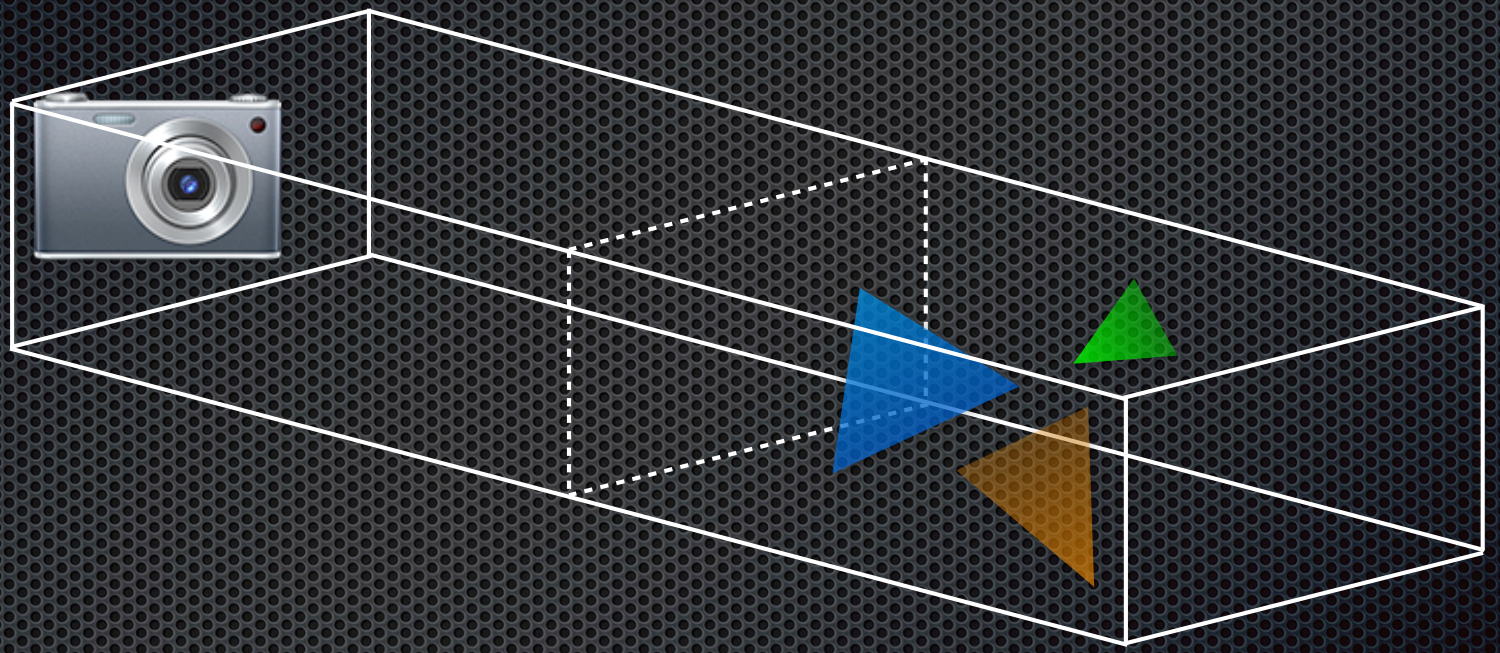
```
glUseProgram(_program) // positions array has 9 floats between -1.0 -> 1.0
glVertexAttribPointer(10, 3, GLenum(GL_FLOAT), GLboolean(FALSE), 0, positions)
glVertexAttribPointer(11, 3, GLenum(GL_FLOAT), GLboolean(FALSE), 0, texCoords)
let modelViewLocation: GLint = (glGetUniformLocation(_program, "modelView")
glUniformMatrix4fv(modelViewLocation, 1, GLboolean(FALSE), modelView)
glBindTexture(GLenum(GL_TEXTURE_2D), _woodTextureInfo.name)
glDrawArrays(GL_TRIANGLES, 0, 3) // texCoords array has 6 floats 0.0 -> 1.0
```



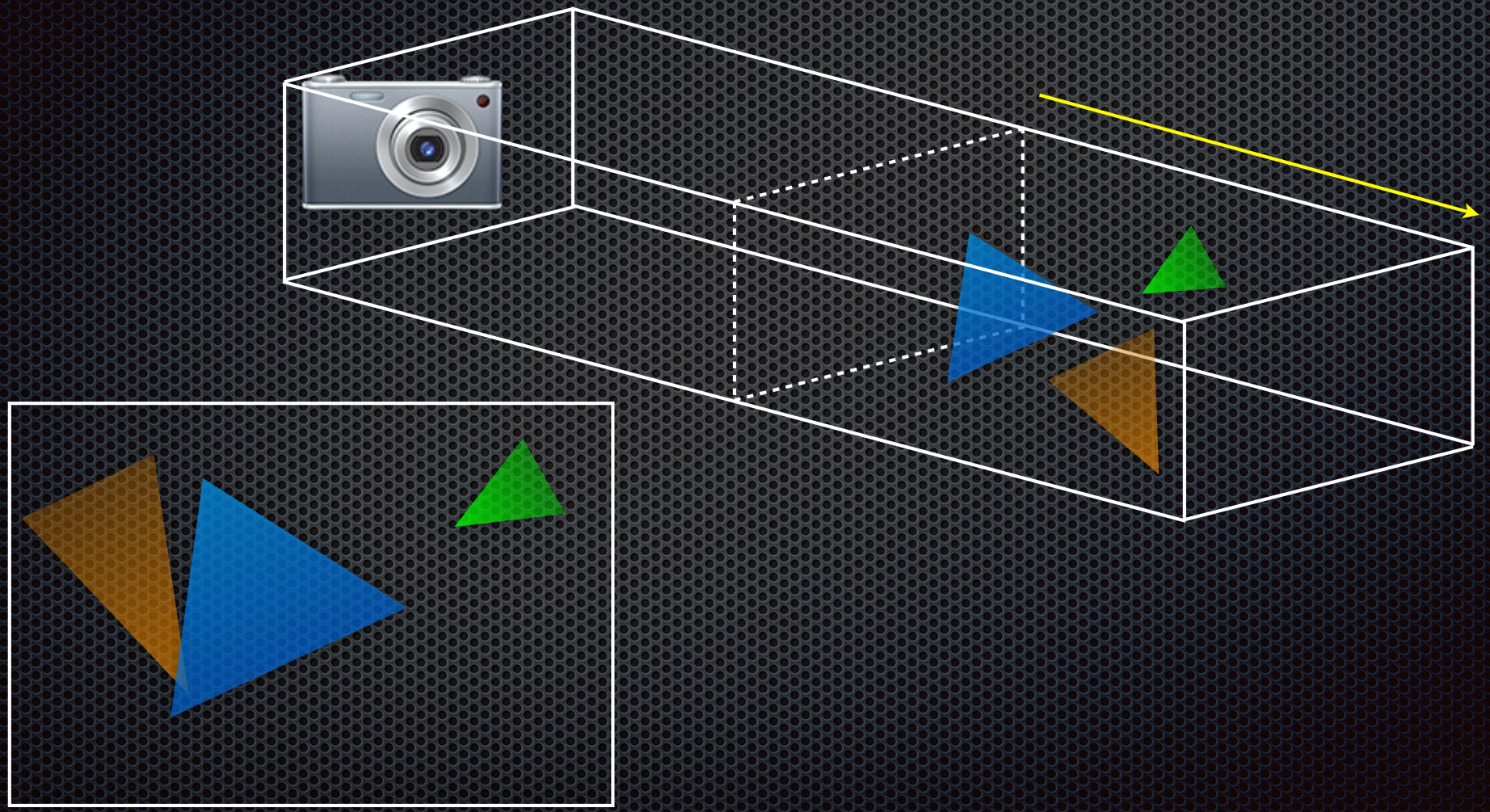
```
uniform mat4 modelView;
attribute vec3 position;
attribute vec2 textureCoordinate;
varying vec2 textureCoordinateInterpolated;
void main()
{
    gl_Position = modelView * vec4(position, 1.0)
    textureCoordinateInterpolated = textureCoordinate; // interpolated
}
```

```
varying highp vec2 textureCoordinateInterpolated;
uniform sampler2D textureUnit;
void main()
{
    gl_FragColor = texture2D(textureUnit, textureCoordinateInterpolated);
}
```

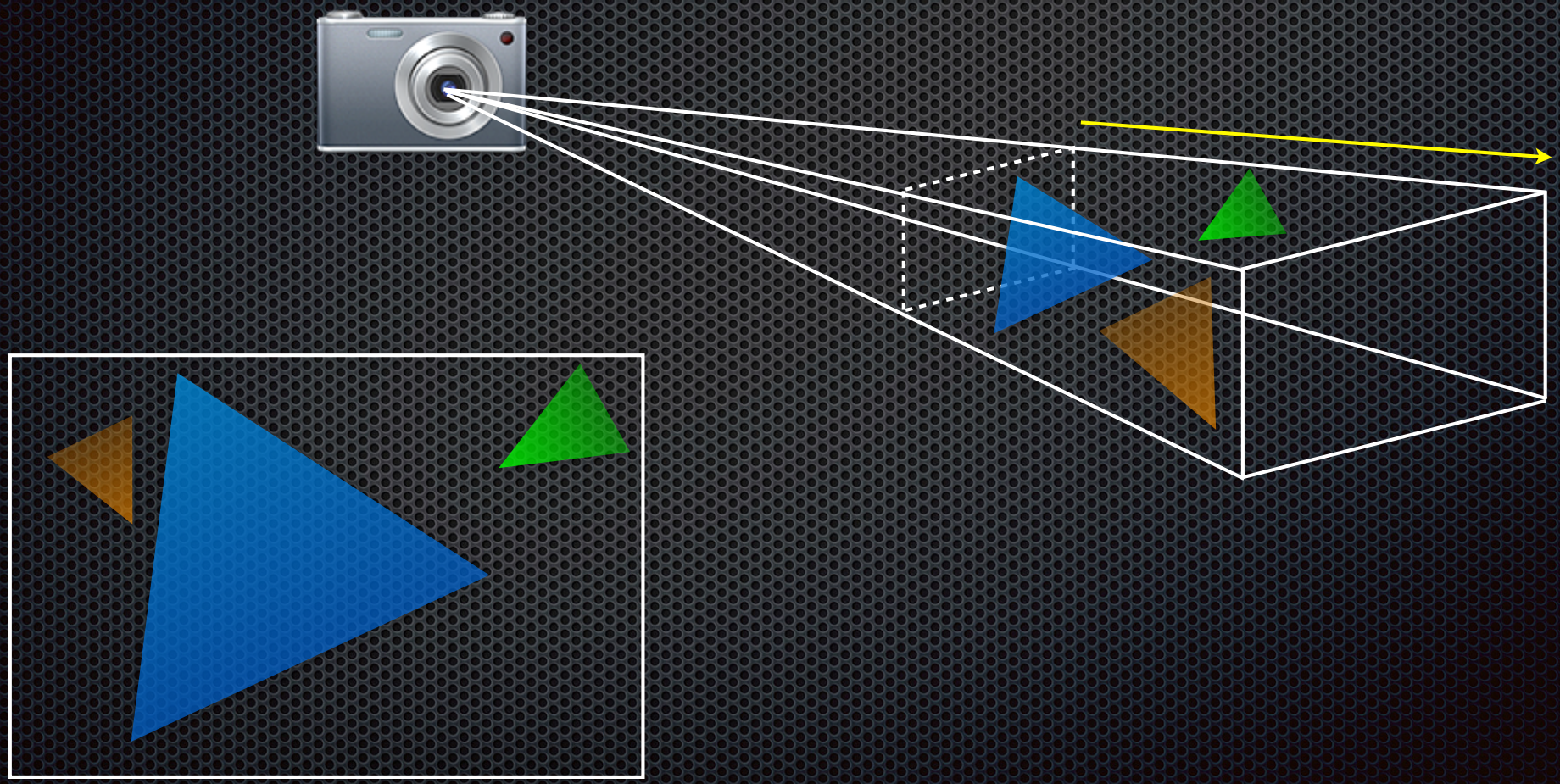
Projection Transformations



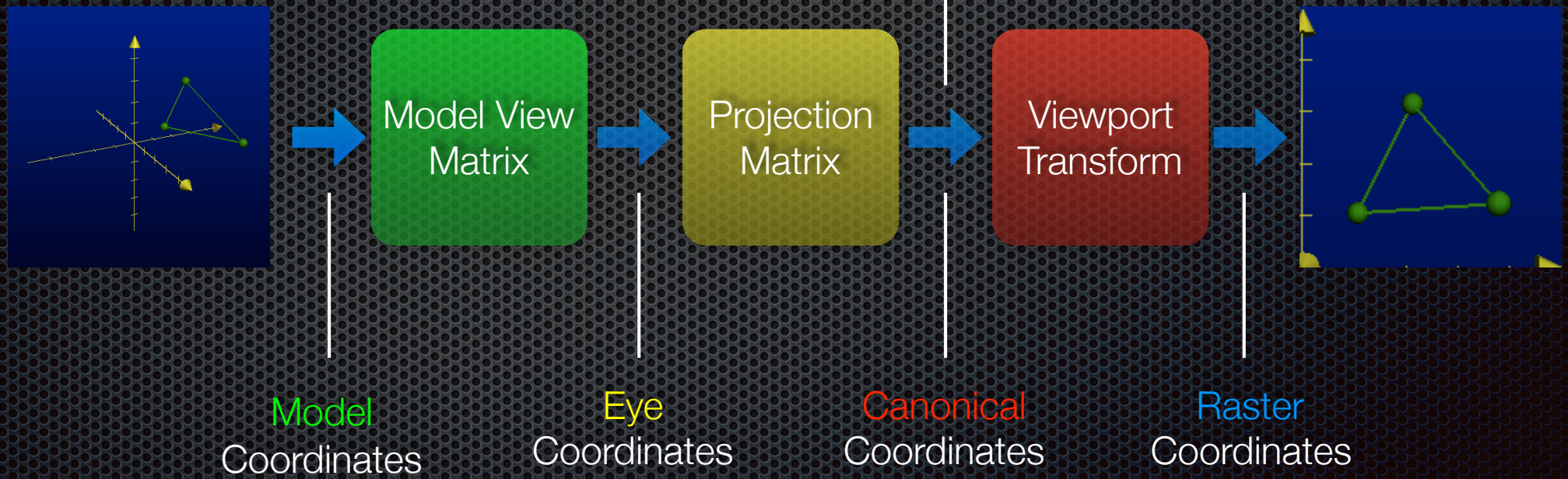
Projection - Orthographic



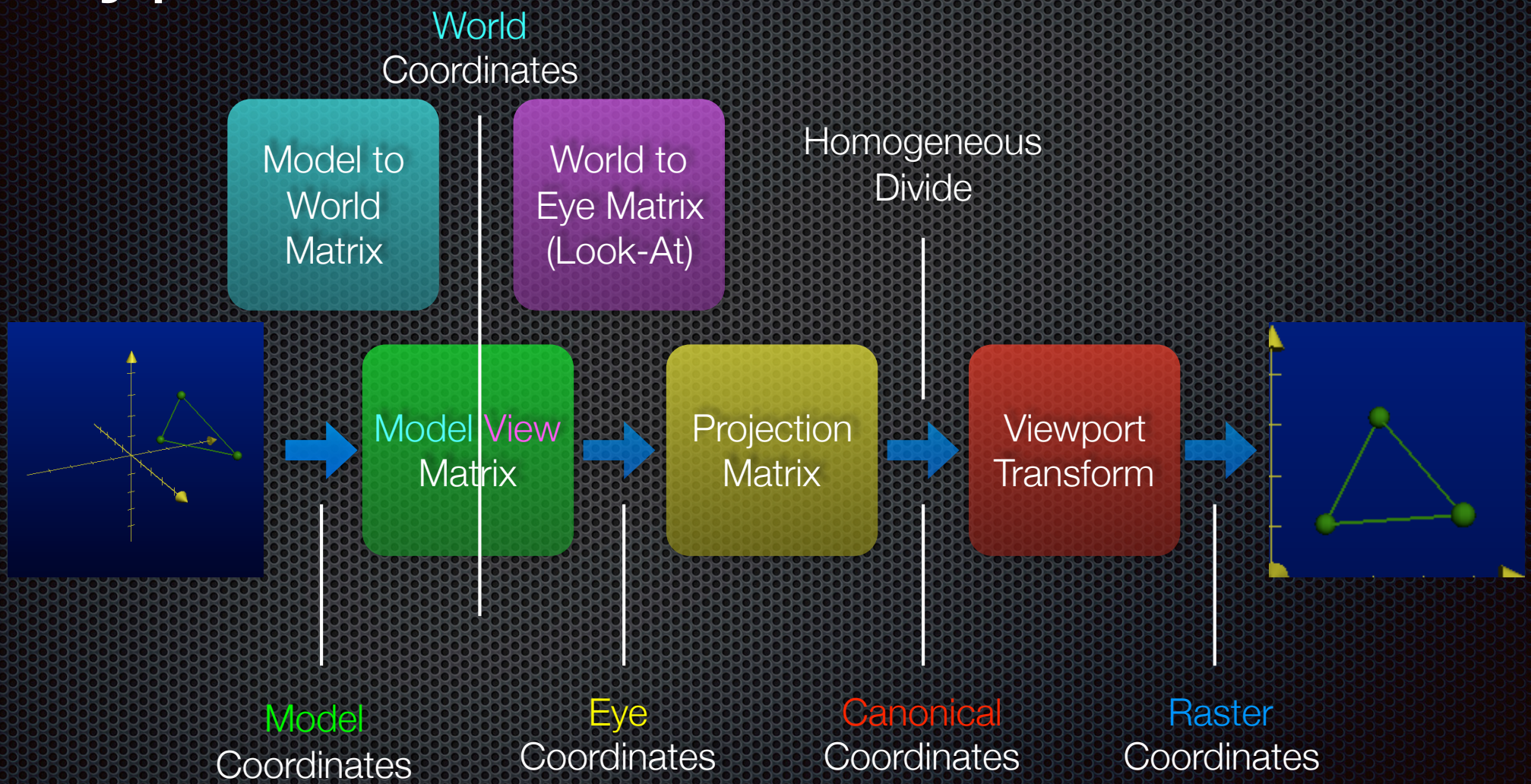
Projection - Perspective



Typical Matrices



Typical Matrices



Model Transformations

Model to World Matrix

Translation

$$T = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and } T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -x \\ 0 & 1 & 0 & -y \\ 0 & 0 & 1 & -z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scale

$$S = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and } S^{-1} = \begin{bmatrix} \frac{1}{x} & 0 & 0 & 0 \\ 0 & \frac{1}{y} & 0 & 0 \\ 0 & 0 & \frac{1}{z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation

X-Rotation

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos a & -\sin a & 0 \\ 0 & \sin a & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Y-Rotation

$$\begin{bmatrix} \cos a & 0 & \sin a & 0 \\ 0 & 1 & 0 & 0 \\ -\sin a & 0 & \cos a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Z-Rotation

$$\begin{bmatrix} \cos a & -\sin a & 0 & 0 \\ \sin a & \cos a & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Let v = (x, y, z)T, and u = v/||v|| = (x', y', z')T.

Also let

$$S = \begin{bmatrix} 0 & -z' & y' \\ z' & 0 & -x' \\ -y' & x' & 0 \end{bmatrix} \quad \text{and } M = uu^T + (\cos a)\,(I - uu^T) + (\sin a)\,S$$

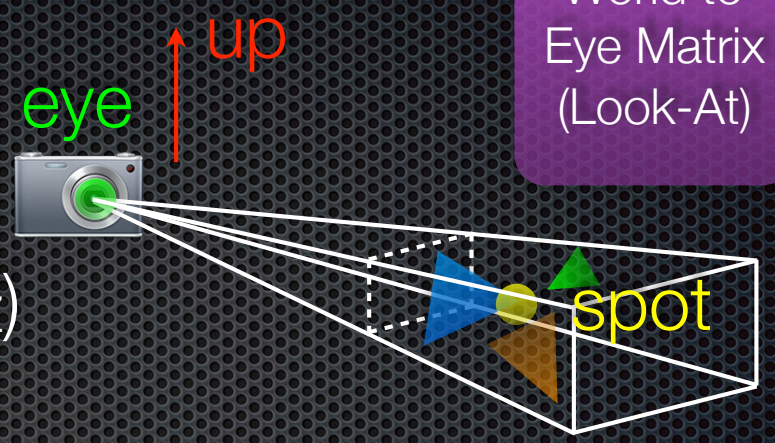
Then

$$R = \begin{bmatrix} m & m & m & 0 \\ m & m & m & 0 \\ m & m & m & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

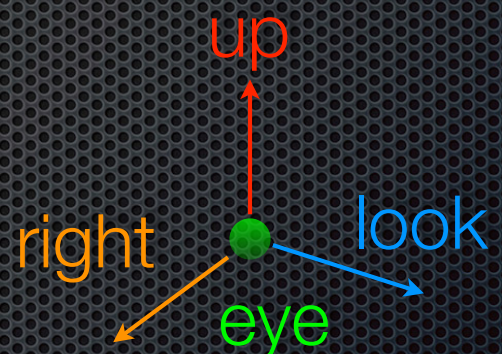
where *m* represents elements from M, which is a 3x3 matrix.

Camera Matrix

World to Eye Matrix (Look-At)



- User positioned at (eye_x,eye_y,eye_z)
- User looks at ($spot_x$, $spot_y$, $spot_z$)
- Any direction can be "up"
- So, user looks down look vector with an implicit direction right



$$\begin{bmatrix} r_x & r_y & r_z & 0 \\ u_x & u_y & u_z & 0 \\ l_x & l_y & l_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

look = spot - eye

right = look x up

up = right x look

look, right, up are normalized

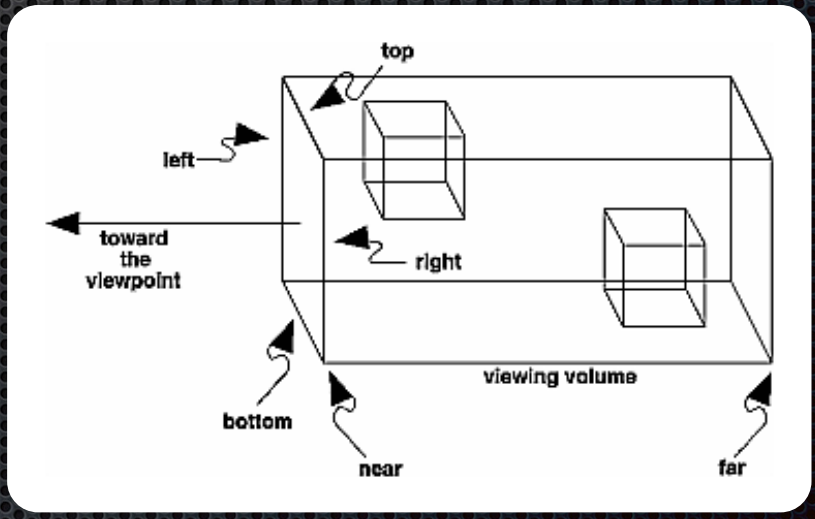
Orthographic Projection

Projection Matrix

Combination Scale & Translation

ortho(l,r,b,t,n,f)

$$R = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } R^{-1} = \begin{bmatrix} \frac{r-l}{2} & 0 & 0 & \frac{r+l}{2} \\ 0 & \frac{t-b}{2} & 0 & \frac{t+b}{2} \\ 0 & 0 & \frac{f-n}{-2} & \frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

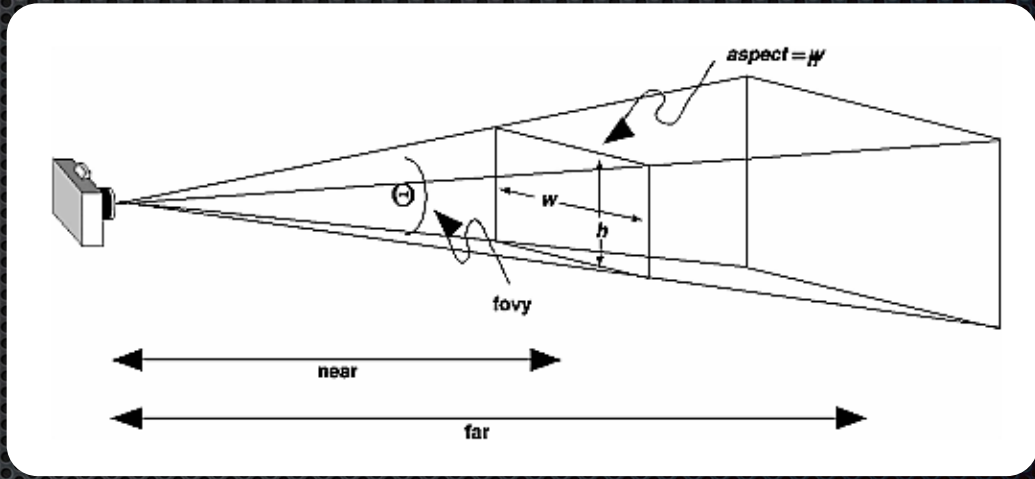


Perspective Projection

Projection Matrix

frustum(l,r,b,t,n,f)

$$R = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad \text{and } R^{-1} = \begin{bmatrix} \frac{r-l}{2n} & 0 & 0 & \frac{r+l}{2n} \\ 0 & \frac{t-b}{2n} & 0 & \frac{t+b}{2n} \\ 0 & 0 & 0 & -1 \\ 0 & 0 & \frac{-(f-n)}{2fn} & \frac{f+n}{2fn} \end{bmatrix}$$




```
func perspective(fovY: Float, aspectRatio: Float, zNear: Float, zFar: Float)
{
    let yMax: Float = zNear * tan(fovY * Float(M_PI / 360.0))
    let yMin: Float = -yMax
    let xMin: Float = yMin * aspectRatio
    let xMax: Float = yMax * aspectRatio

    frustum(xMin, xMax, yMin, yMax, zNear, zFar)
}
```

Viewport Transform

Viewport Transform

Viewport Transform as a Matrix

$$R = \begin{bmatrix} \frac{w}{2} & 0 & 0 & \frac{w}{2} + x \\ 0 & \frac{h}{2} & 0 & \frac{h}{2} + y \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Discards Z Coordinate!

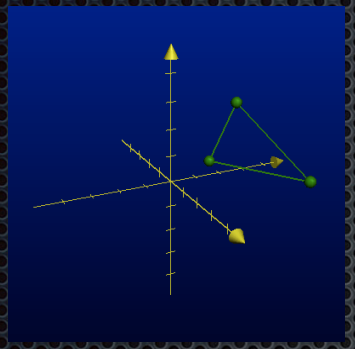
Typical Matrices

World
Coordinates

$$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

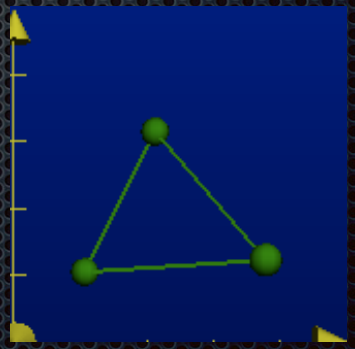
$$\begin{bmatrix} r_x & r_y & r_z & 0 \\ u_x & u_y & u_z & 0 \\ l_x & l_y & l_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Homogeneous
Divide



$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \frac{w}{2} & 0 & 0 & \frac{w}{2}+x \\ 0 & \frac{h}{2} & 0 & \frac{h}{2}+y \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



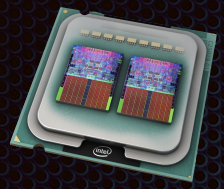
Model
Coordinates

Eye
Coordinates

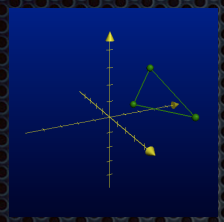
Canonical
Coordinates

Raster
Coordinates

X: -1→1 Y: -1→1 Z: -1→1



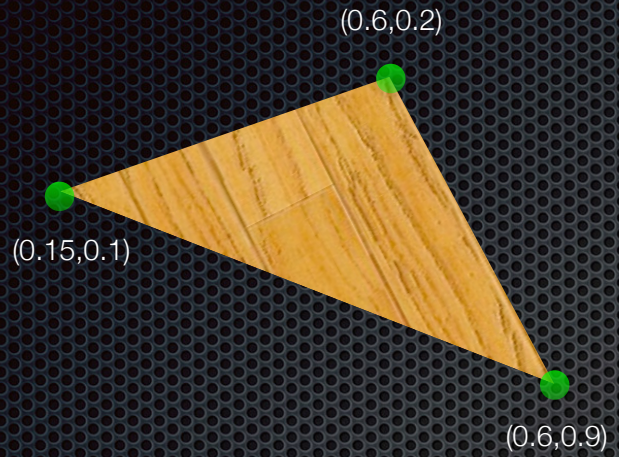
```
glUseProgram(_program) // positions array has 9 floats between -1.0 -> 1.0
glVertexAttribPointer(10, 3, GLenum(GL_FLOAT), GLboolean(FALSE), 0, positions)
glVertexAttribPointer(11, 3, GLenum(GL_FLOAT), GLboolean(FALSE), 0, texCoords)
let modelViewLocation: GLint = (glGetUniformLocation(_program, "modelView")
glUniformMatrix4fv(modelViewLocation, 1, GLboolean(FALSE), modelView)
let projectionLocation: GLint = (glGetUniformLocation(_program, "projection")
glUniformMatrix4fv(projectionLocation, 1, GLboolean(FALSE), projectionMatrix);
glBindTexture(GLenum(GL_TEXTURE_2D), _woodTextureInfo.name)
glDrawArrays(GL_TRIANGLES, 0, 3) // texCoords array has 6 floats 0.0 -> 1.0
```

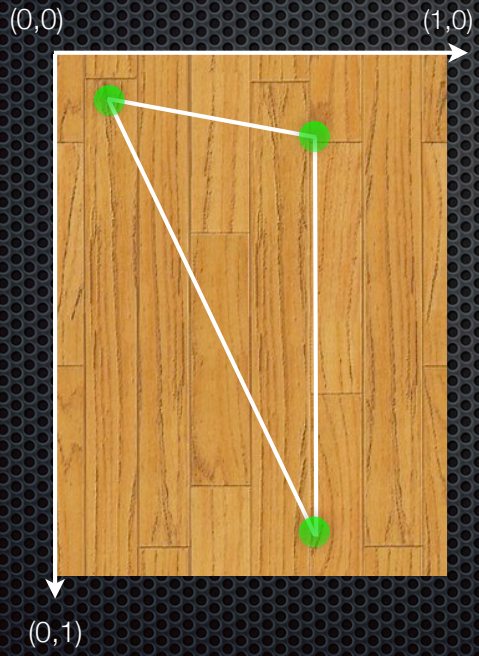


```
attribute vec3 position;
attribute vec2 textureCoordinate;
uniform mat4 modelView;
uniform mat4 projection;
varying vec2 textureCoordinateInterpolated;
void main()
{
    gl_Position = projection * modelView * vec4(position, 1.0)
    textureCoordinateInterpolated = textureCoordinate; // interpolated
}
```

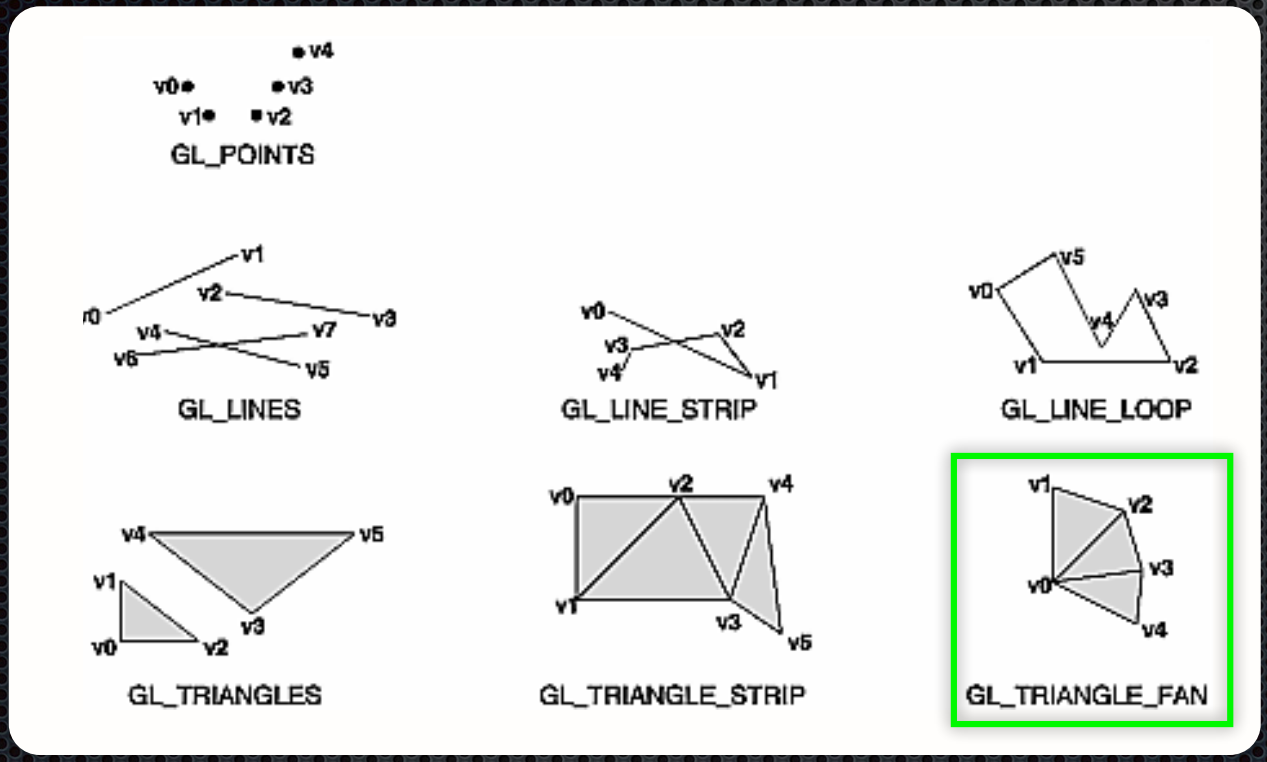
```
varying highp vec2 textureCoordinateInterpolated;
uniform sampler2D textureUnit;
void main()
{
    gl_FragColor = texture2D(textureUnit, textureCoordinateInterpolated);
}
```

Meshes





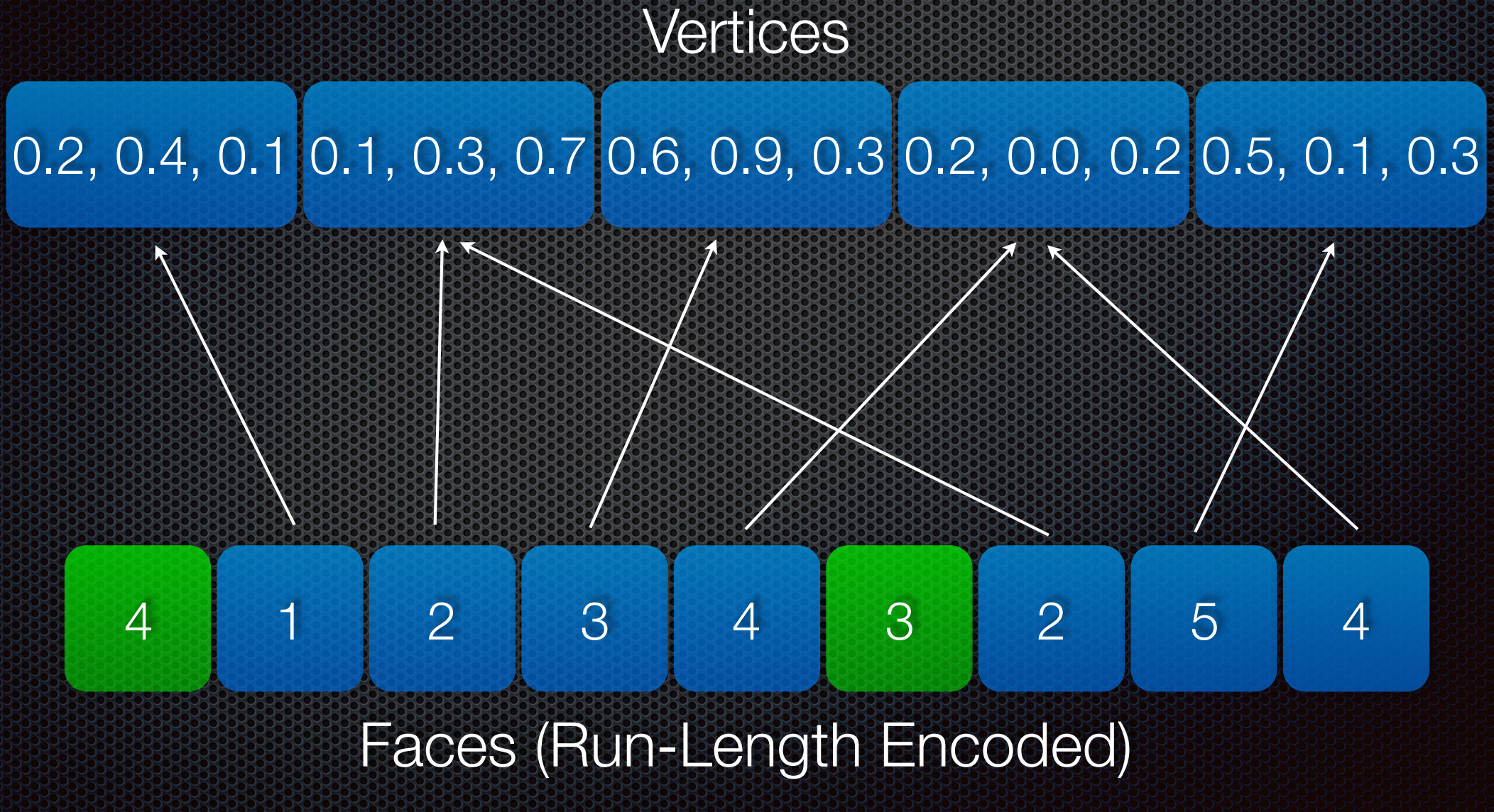
Meshes



Cube.obj

```
# Notes:
# v - vertices in x,y,z coordinates
# vt - texture in u,v coordinates
# vn - normals in nx,ny,nz coordinates
# f - faces: map vertices, UVs, normals

g cube

v 0.0 0.0 0.0
v 0.0 0.0 1.0
v 0.0 1.0 0.0
v 0.0 1.0 1.0
v 1.0 0.0 0.0
v 1.0 0.0 1.0
v 1.0 1.0 0.0
v 1.0 1.0 1.0

vt 0.0 0.0
vt 1.0 0.0
vt 1.0 1.0
vt 0.0 1.0

vn 0.0 0.0 1.0
vn 0.0 0.0 -1.0
vn 0.0 1.0 0.0
vn 0.0 -1.0 0.0
vn 1.0 0.0 0.0
vn -1.0 0.0 0.0

f 1/1/2 7/3/2 5/2/2
f 1/1/2 3/4/2 7/3/2
f 1/1/6 4/3/6 3/4/6
f 1/1/6 2/2/6 4/3/6
f 3/1/3 8/3/3 7/4/3
f 3/1/3 4/2/3 8/3/3
f 5/2/5 7/3/5 8/4/5
f 5/2/5 8/4/5 6/1/5
f 1/1/4 5/2/4 6/3/4
f 1/1/4 6/3/4 2/4/4
f 2/1/1 6/2/1 8/3/1
f 2/1/1 8/3/1 4/4/1
```

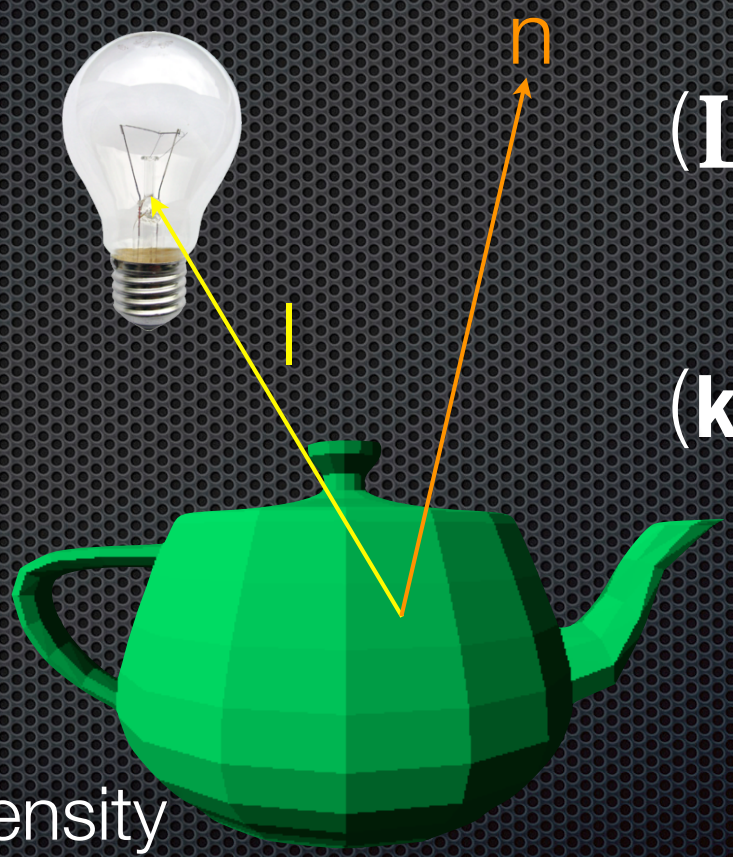
Meshes


Vertices
0.2, 0.4, 0.1
0.1, 0.3, 0.7
0.6, 0.9, 0.3
0.2, 0.0, 0.2
0.5, 0.1, 0.3
4
1
2
3
4
3
2
5
4
Faces (Run-Length Encoded)

Diffuse Lighting

$$I_{dr} = L_{dr} k_{dr} \hat{l} \bullet \hat{n}$$
$$I_{dg} = L_{dg} k_{dg} \hat{l} \bullet \hat{n}$$
$$I_{db} = L_{db} k_{db} \hat{l} \bullet \hat{n}$$

$$0 < I_{dx} < 1$$

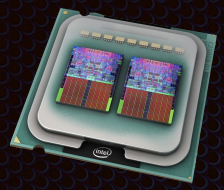


(L_{dr}, L_{dg}, L_{db}) = light color

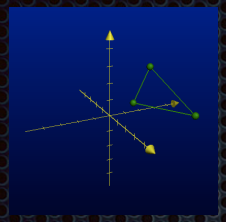
(k_{dr}, k_{dg}, k_{db}) = material color

L_{dx} = light channel intensity

k_{dx} = material diffuse coefficient for channel



```
/* Supply position, normal, and textureCoordinate arrays. Set model view and projection matrices. */
/* Give the location of the light in eye coordinates (apply the model view matrix to it). */
/* Also define the color (and implicitly the brightness) of the light. This can exceed 1 for effect! */
```



```
attribute vec3 position;
attribute vec3 normal;
attribute vec2 textureCoordinate;

uniform mat4 modelView;
uniform mat4 projection;

varying highp vec4 positionEyeCoordinates;
varying highp vec3 normalEyeCoordinates;
varying highp vec2 textureCoordinateInterpolated;

void main()
{
    positionEyeCoordinates = modelView * vec4(position, 0.0);
    gl_Position = projection * positionEyeCoordinates;

    normalEyeCoordinates = vec3(normalize(modelView * vec4(normal, 0.0)));
    textureCoordinateVarying = textureCoordinate;
}
```

```
uniform sampler2D textureUnit;
uniform vec4 lightPositionEyeCoordinates;
uniform vec4 lightColor;

varying highp vec4 positionEyeCoordinates;
varying highp vec3 normalEyeCoordinates;
varying highp vec2 textureCoordinateInterpolated;

void main()
{
    vec3 incident = vec3(normalize(lightPositionEyeCoordinates - positionEyeCoordinates));
    float incidence = max(0.0, dot(incidentVector, normalEyeCoordinates));
    gl_FragColor = texture2D(textureUnit, textureCoordinateInterpolated) * lightColor * incidence;
}
```

iOS Implementation

- Custom Shaders
  - Performant
  - Low-level control
  - Custom effects
  - Porting-friendly
  - Easier integration with existing OpenGL code
- GLKBaseEffect
  - Easy to use
  - Has own shaders
  - Defined capabilities
  - OOP Structured input data
  - Vector / Matrix structures