CSE 5542 - Real Time Rendering
Week 3 & 4

THE OHIO STATE UNIVERSITY

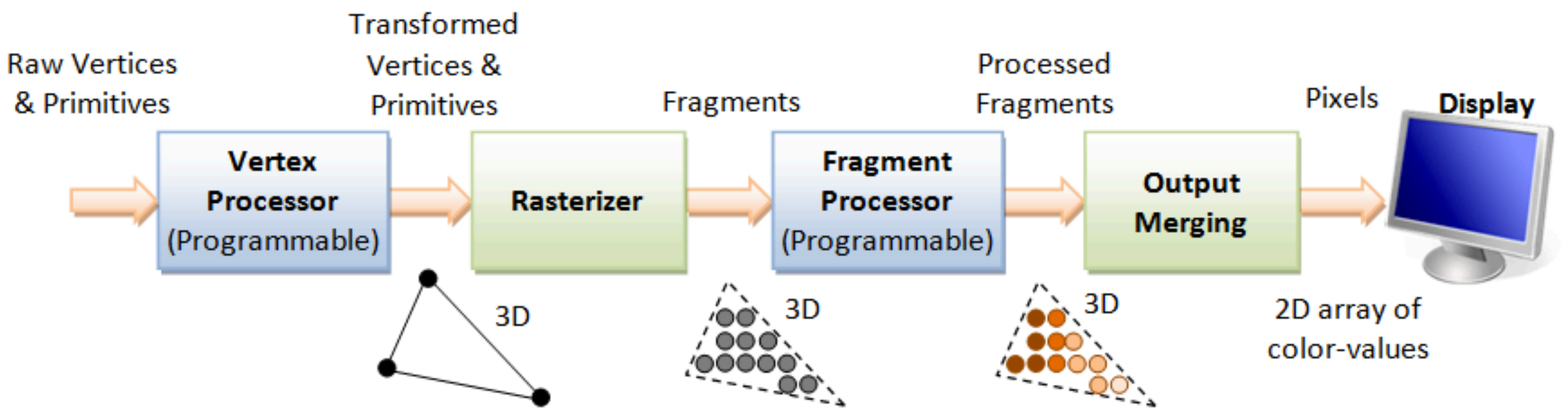
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Slides(Mostly) Courtesy – E. Angel and D. Shreiner

Program Execution

- WebGL runs in browser
  - complex interaction with OS, Window system, browser, & code (HTML and JS)
- Simple model
  - Start with HTML file
  - files read in asynchronously
  - start with onload function
    - event driven input

Hardware Rendering Pipeline



Raw Vertices & Primitives
Vertex Processor (Programmable)
Transformed Vertices & Primitives
Rasterizer
3D
Fragments
Fragment Processor (Programmable)
3D
Processed Fragments
Output Merging
3D
Pixels
Display
2D array of color-values

$$\begin{tikzcd}\boxed{\text{host interface}} \arrow[r] & \boxed{\text{vertex processing}} \arrow[r] & \boxed{\text{triangle setup}} \arrow[r] & \boxed{\text{pixel processing}} \arrow[r] & \boxed{\text{memory interface}}\end{tikzcd}$$

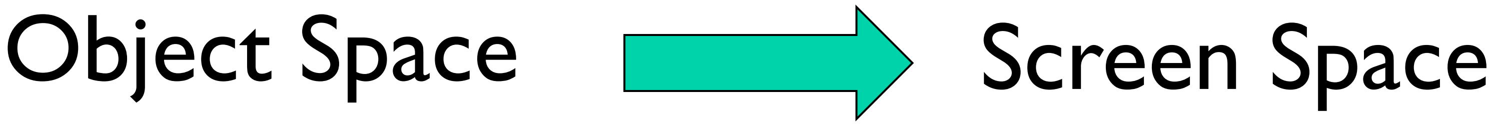
Space ?

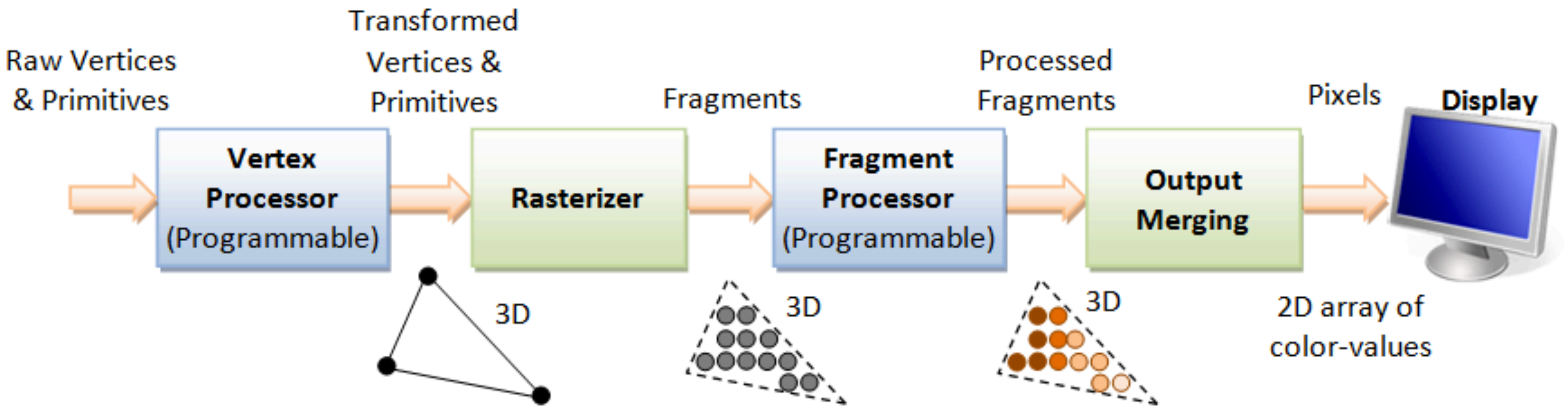
```
point2 vertices[3] = {point2(0.0, 0.0),
     point2( 0.0, 1.0), point2(1.0, 1.0)};
```

Coordinate Systems

- Units in **points** - *object, world, model problem coordinates*
- Viewing specifications are also in object coordinates
- Same for lights
- Eventually pixels will be produced in *window coordinates*
- WebGL - internal representations not visible to application but important in shaders

Transform Spaces




Raw Vertices & Primitives
Vertex Processor (Programmable)
Transformed Vertices & Primitives
Rasterizer
Fragments
Fragment Processor (Programmable)
Processed Fragments
Output Merging
Pixels
Display
3D
3D
3D
2D array of color-values

Most important is *clip coordinates*

Shaders – Clip Space

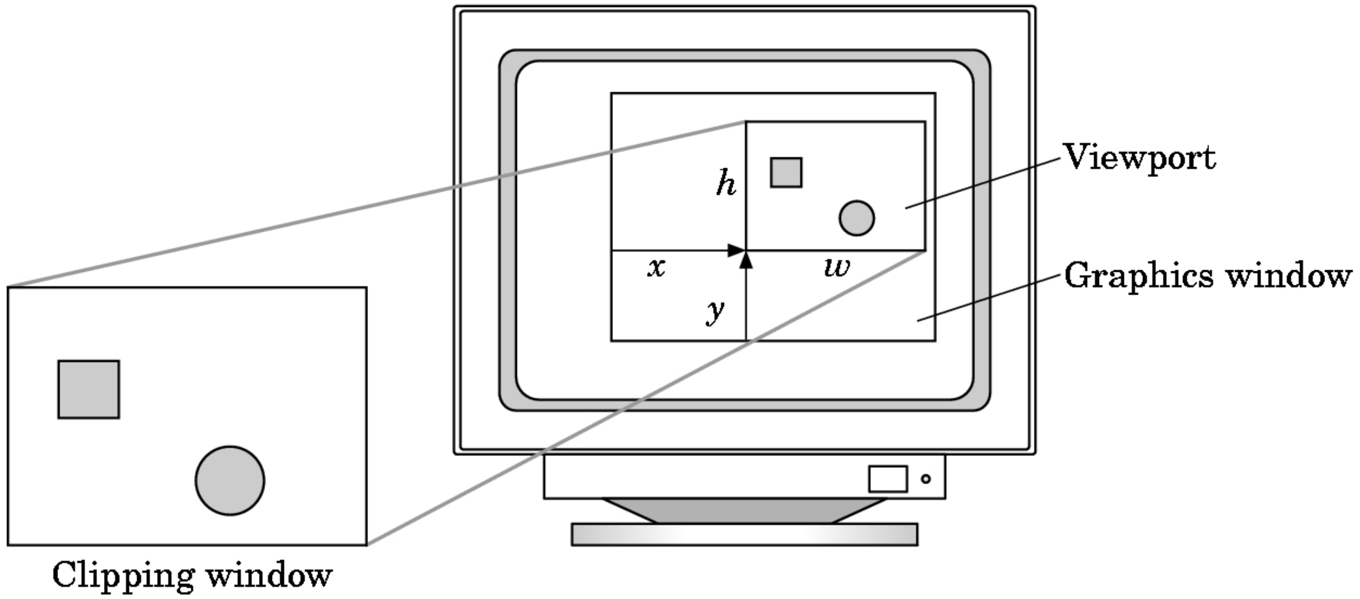
- Vertex shader must output in clip coordinates
- Input to fragment shader from rasterizer is in window coordinates
- Application can provide vertex data in any coordinate system
- Shader must eventually produce gl_Position in clip coordinates

Viewports

- Not use entire window for image:

  gl.viewport(x,y,w,h)

- Values in pixels (window coordinates)

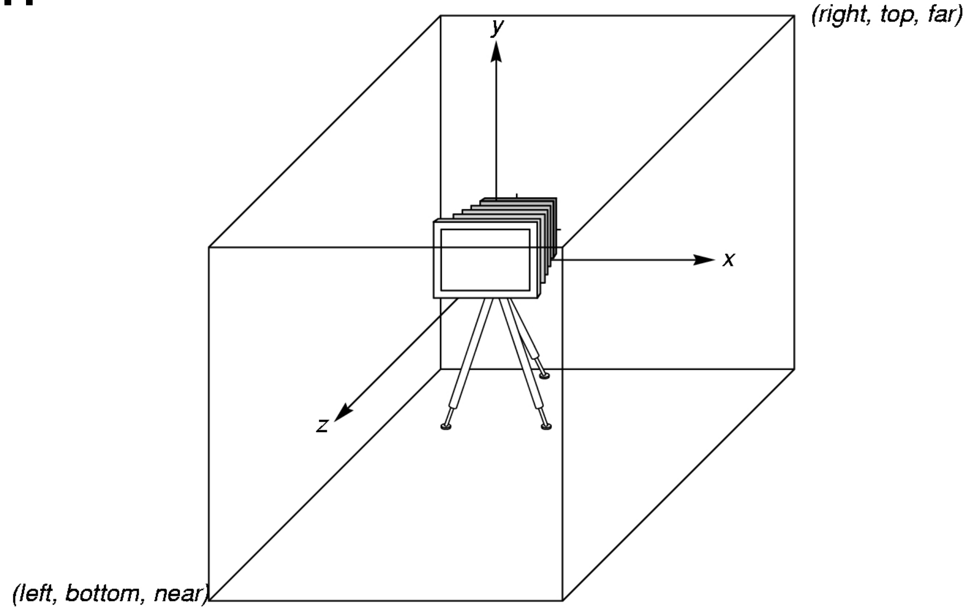


WebGL Camera

- Camera at origin in object space pointing in -z direction

- Default viewing volume

  - box centered at origin with sides of length 2

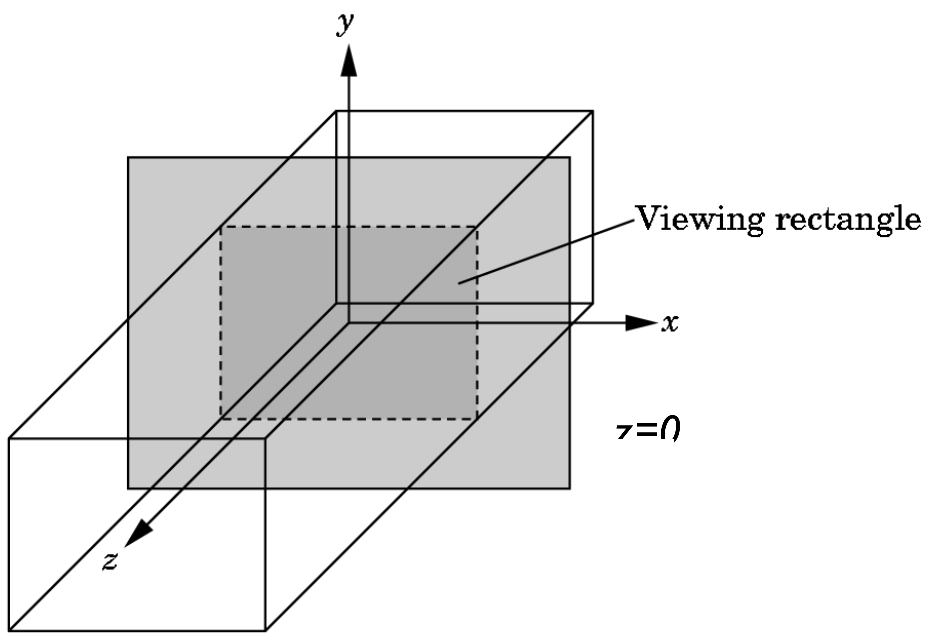


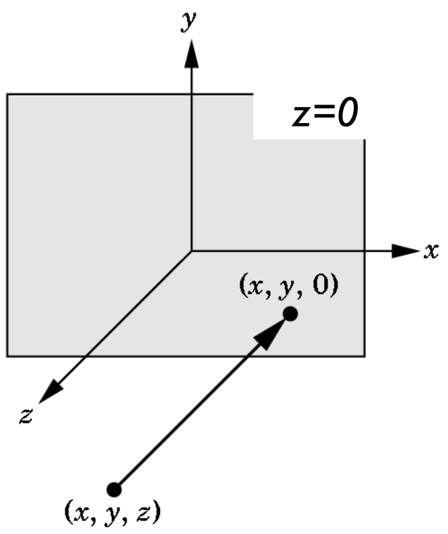
Transformations & Viewing

- WebGL - projection with matrix (transformation) before rasterization
- Pre 3.1 OpenGL- transformation functions which have been deprecated
  - glRotate
  - glTranslate
  - …
- Three choices in WebGL
  - Application code
  - GLSL functions
  - MV.js

Orthographic Viewing

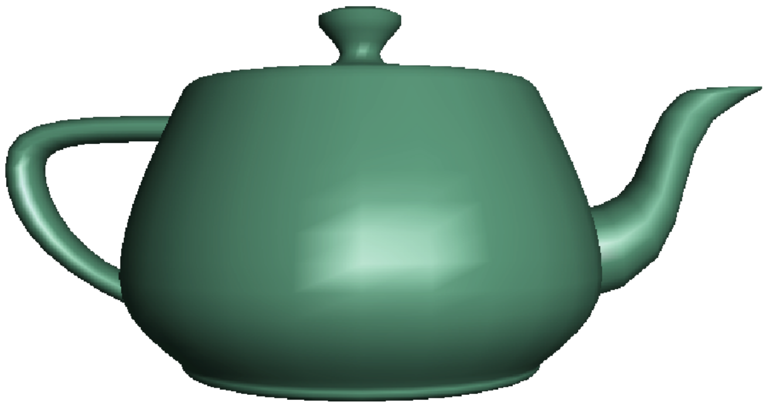
Points projected forward along *z* axis onto plane *z=0*



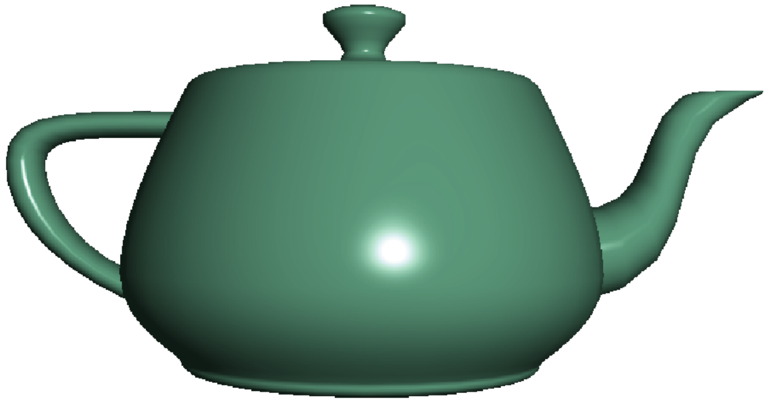


Shaders

Fragment vs Vertex Shader



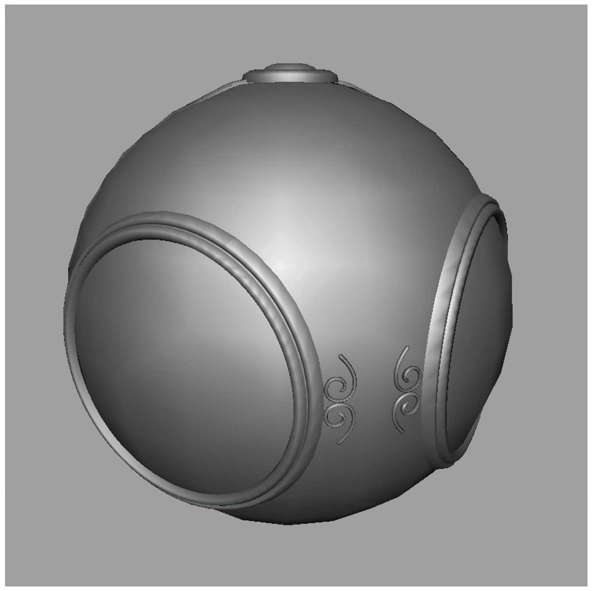
per vertex lighting

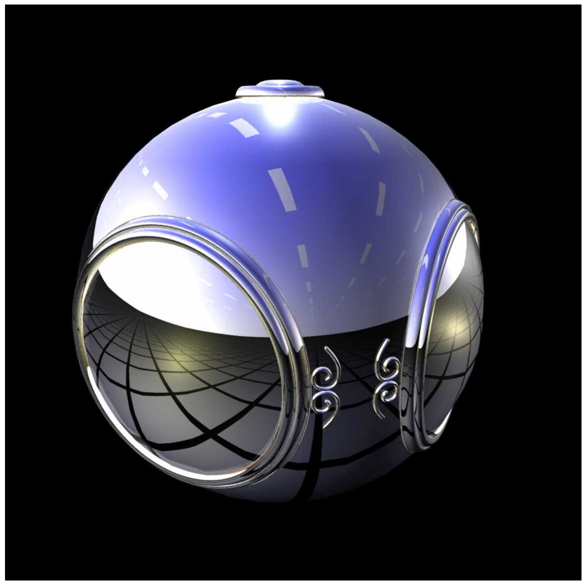


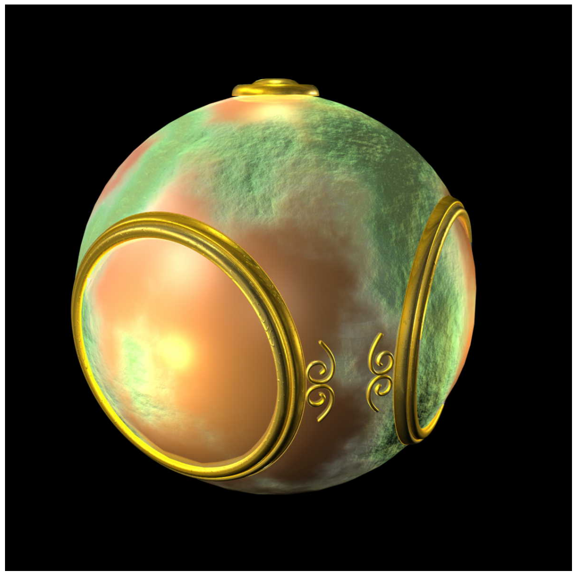
per fragment lighting

Fragment Shader Applications

Texture mapping


smooth shading


environment mapping


bump mapping

Other Vertex Shaders

- Moving vertices
  - Morphing
  - Wave motion
  - Fractals
- Lighting
  - More realistic models
  - Cartoon shaders

Shader Evolution

- First programmable shaders were programmed in an assembly-like manner
- OpenGL extensions added functions for vertex and fragment shaders
- Cg (C for graphics) C-like language for programming shaders
  - Works with both OpenGL and DirectX
  - Interface to OpenGL complex
- OpenGL Shading Language (GLSL)

OpenGL and GLSL

- Shader based OpenGL is based less on a state machine model than a data flow model
- Most state variables, attributes and related pre 3.1 OpenGL functions have been deprecated
- Action happens in shaders
- Job is application is to get data to GPU

GLSL

- OpenGL Shading Language
- Part of OpenGL 2.0 and up
- High level C-like language
- New data types
  - Matrices
  - Vectors
  - Samplers

GLSL

- C-like with
  - Matrix and vector types (2, 3, 4 dimensional)
  - Overloaded operators
  - C++ like constructors
- Similar to Nvidia's Cg and Microsoft HLSL
- Code sent to shaders as source code
- As of OpenGL 3.1, application must provide shaders

Simple Vertex Shader

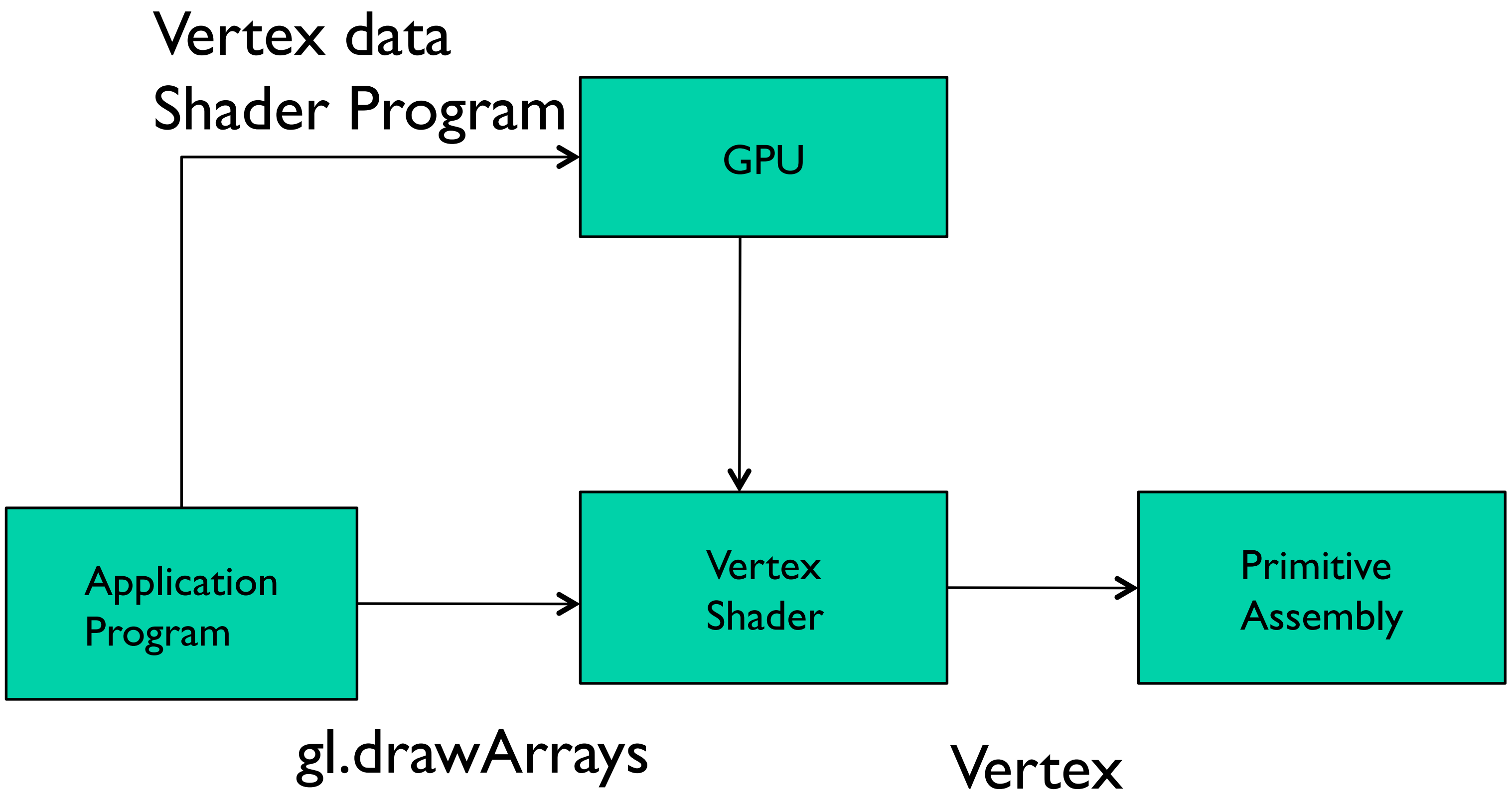
input from application

```
attribute vec4 vPosition;
void main(void)
{
    gl_Position = vPosition;
}
```

must link to variable in application

built in variable

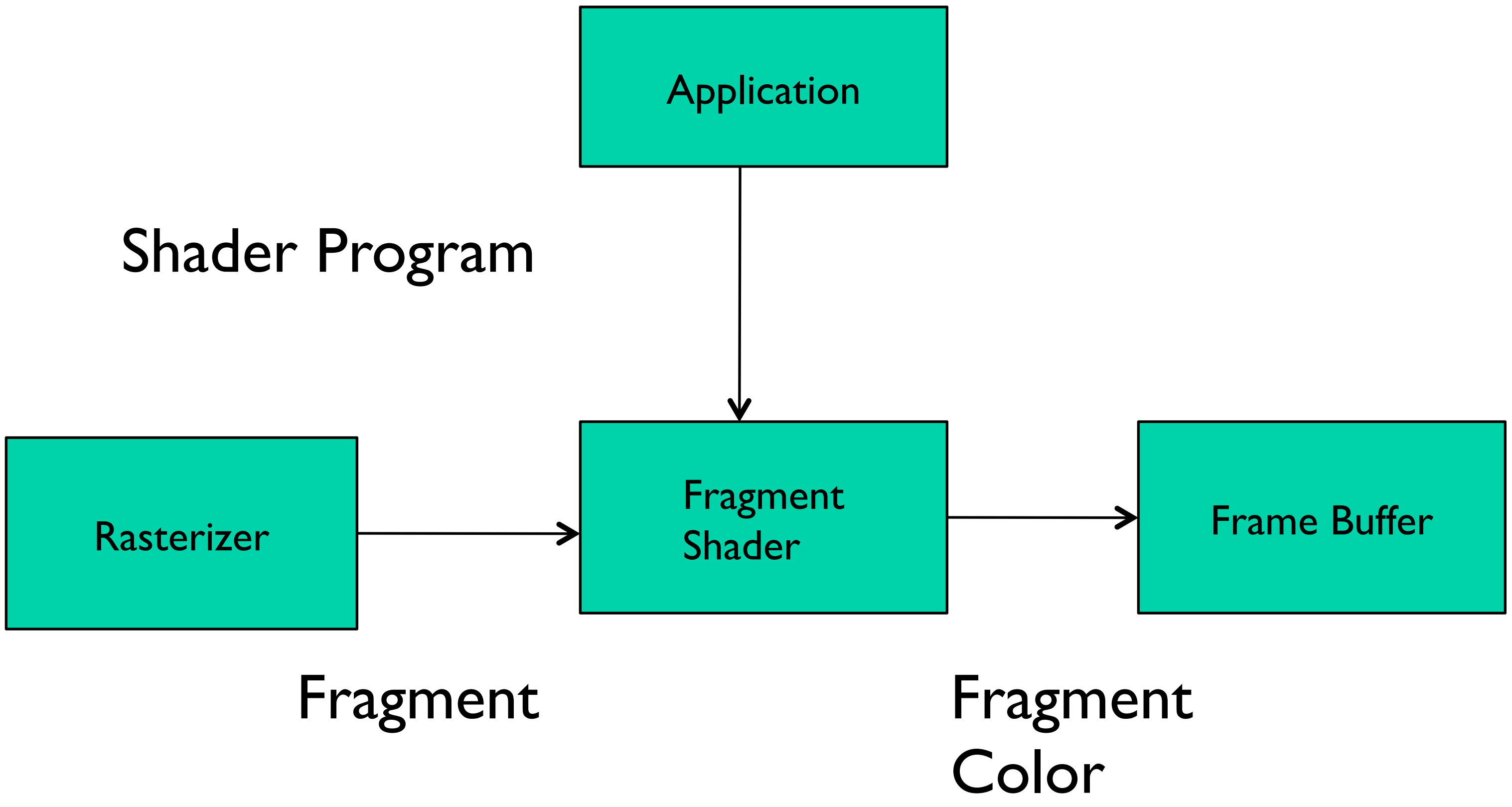
Execution Model



Simple Fragment Program

```
precision mediump float;
void main(void)
{
   gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Execution Model

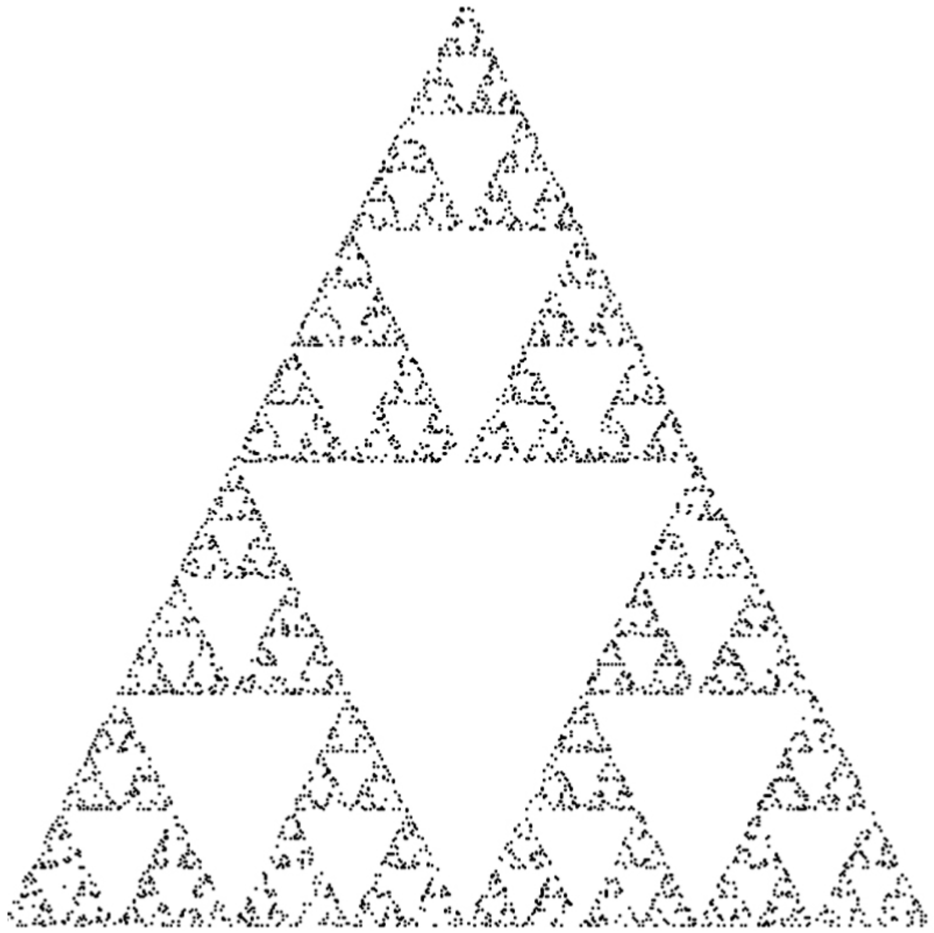


Still Maximal Portability

- Display device independent
- Window system independent
- Operating system independent

Eine Example

The Sierpinski Gasket

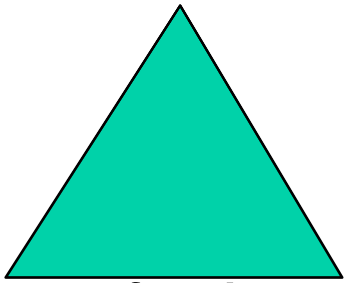


init()

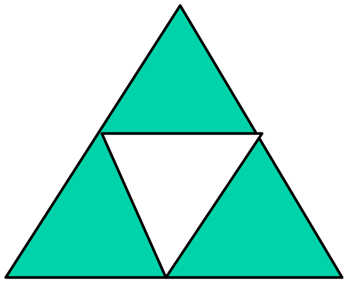
```
var program = initShaders( gl, "vertex-shader", "fragment-shader" );
    gl.useProgram( program );
   …
```

Sierpinski Gasket (2D)

- Start with a triangle



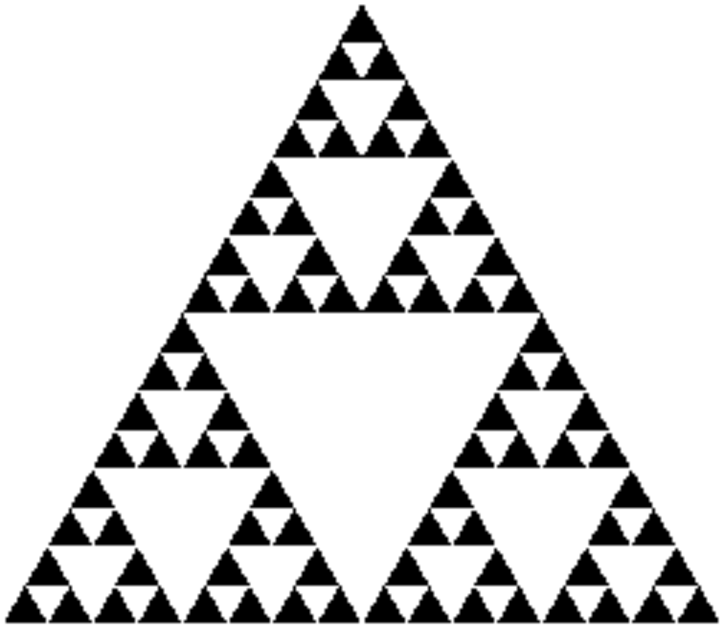
- Connect bisectors of sides and remove central triangle



- Repeat

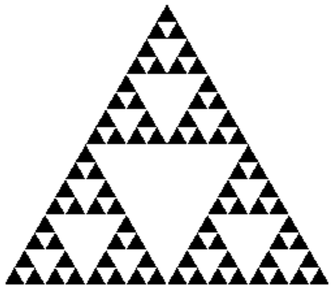
Example

Five subdivisions



Gasket == fractal

- Consider the filled area (black) and the perimeter (the length of all the lines around the filled triangles)
- As we continue subdividing
  - the area goes to zero
  - but the perimeter goes to infinity
- This is not an ordinary geometric object
  - It is neither two- nor three-dimensional
- It is a *fractal* (fractional dimension) object



Example

http://www.cs.unm.edu/~angel/WebGL/7E/02/gasket2.html

http://www.cs.unm.edu/~angel/WebGL/7E/02/gasket2.js

Gasket Program

- HTML file
  - Same as in other examples
  - Pass through vertex shader
  - Fragment shader sets color
  - Read in JS file

Gasket Program

```
var points = [];
var NumTimesToSubdivide = 5;

/* initial triangle */

 var vertices = [
        vec2( -1, -1 ),
        vec2(  0,  1 ),
        vec2(  1, -1 )
    ];

divideTriangle( vertices[0],vertices[1],
     vertices[2], NumTimesToSubdivide);
```

Draw one triangle

```
/* display one triangle  */

function triangle( a, b, c ){
    points.push( a, b, c );
}
```

Triangle Subdivision

```
function divideTriangle( a, b, c, count ){
 // check for end of recursion
    if ( count === 0 ) {
    triangle( a, b, c );
    }
    else {
//bisect the sides
    var ab = mix( a, b, 0.5 );
    var ac = mix( a, c, 0.5 );
    var bc = mix( b, c, 0.5 );
    --count;
// three new triangles
    divideTriangle( a, ab, ac, count-1 );
    divideTriangle( c, ac, bc, count-1 );
    divideTriangle( b, bc, ab, count-1 );
    }
}
```

init()

```
var program = initShaders( gl, "vertex-shader", "fragment-
    shader" );
    gl.useProgram( program );
    var bufferId = gl.createBuffer();
    gl.bindBuffer( gl.ARRAY_BUFFER, bufferId )
    gl.bufferData( gl.ARRAY_BUFFER, flatten(points),
    gl.STATIC_DRAW );
var vPosition = gl.getAttribLocation( program, "vPosition" );
    gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
    gl.enableVertexAttribArray( vPosition );
    render();
```

Render Function

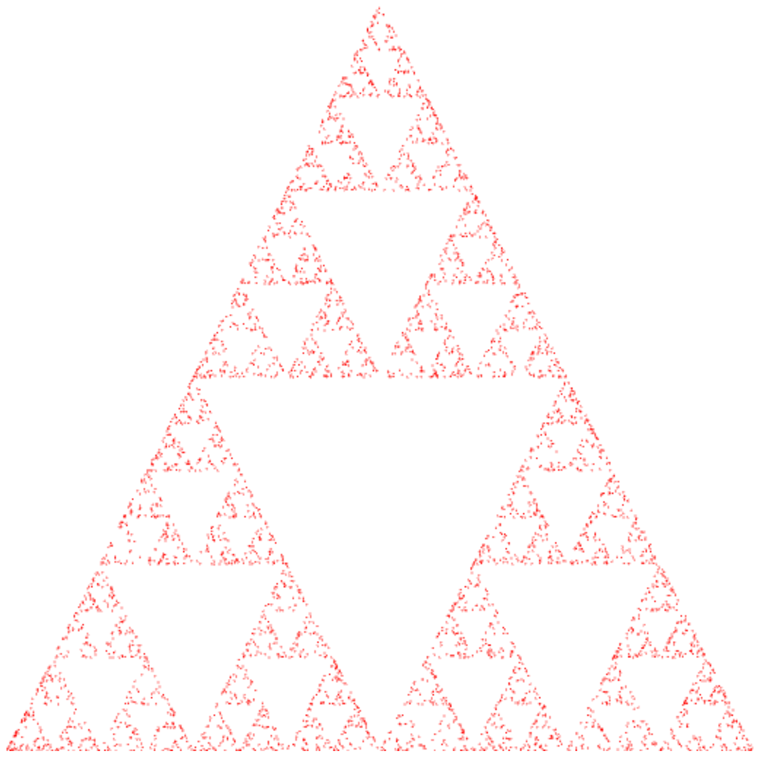
```
function render(){
    gl.clear( gl.COLOR_BUFFER_BIT );
    gl.drawArrays( gl.TRIANGLES, 0, points.length );
}
```

Shaders

```
attribute vec4 vPosition;
void main()
{
        gl_PointSize = 1.0;
        gl_Position = vPosition;
}

precision mediump float;
void main()
{
   gl_FragColor = vec4( 1.0, 0.0, 0.0, 1.0 );
}
```

Other Examples



http://www.cs.unm.edu/~angel/WebGL/7E/02/gasket1.html

http://www.cs.unm.edu/~angel/WebGL/7E/02/gasket1.js

Similarly

http://www.cs.unm.edu/~angel/WebGL/7E/02/gasket1v2.html

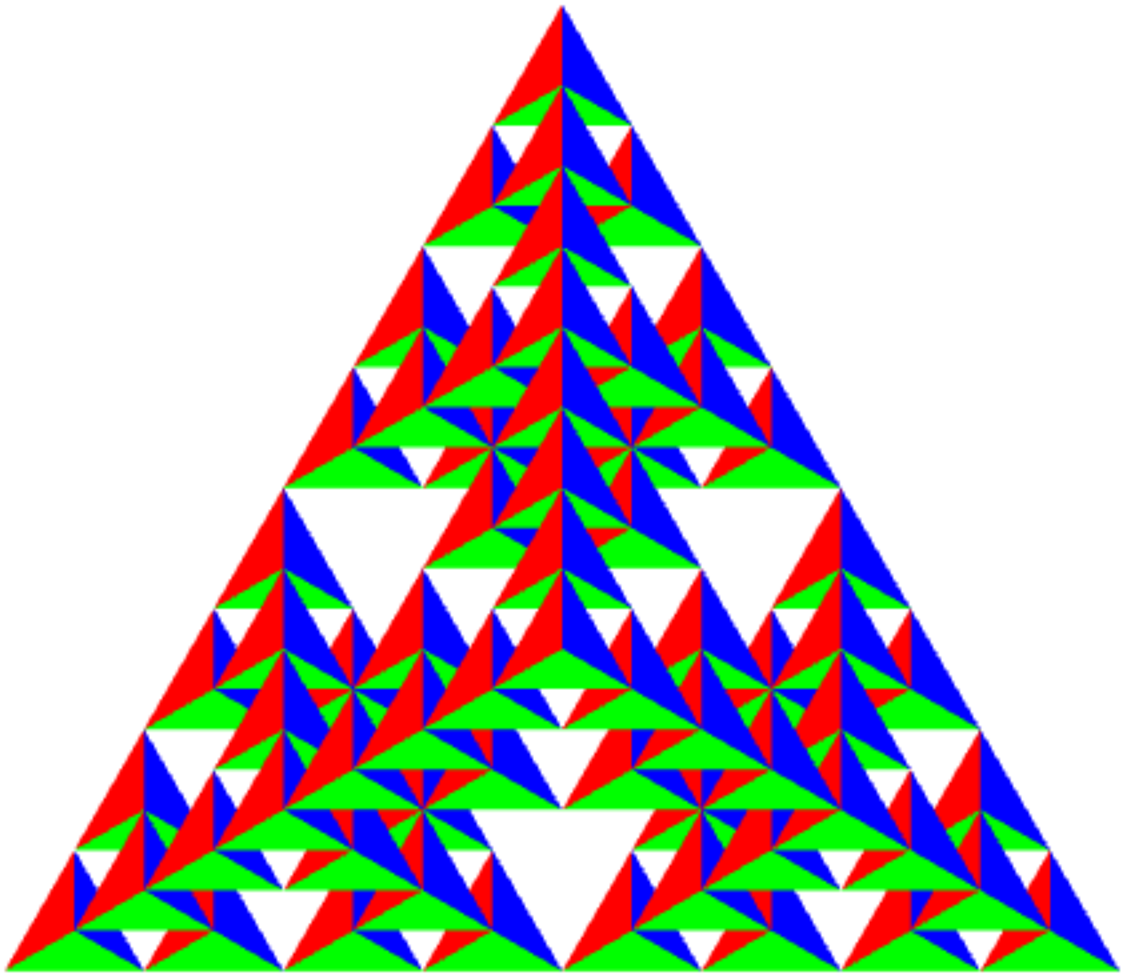
http://www.cs.unm.edu/~angel/WebGL/7E/02/gasket1v2.js

Psychedelic Gaskets

http://www.cs.unm.edu/~angel/WebGL/7E/02/gasket3.html

http://www.cs.unm.edu/~angel/WebGL/7E/02/gasket3.js

3D Gaskets



Many More Examples

http://www.cs.unm.edu/~angel/WebGL/7E/CLASS/

Shaders

```
attribute vec4 vPosition;
void main()
{
	gl_PointSize = 1.0;
	gl_Position = vPosition;
}

precision mediump float;
void main()
{
   gl_FragColor = vec4( 1.0, 0.0, 0.0, 1.0 );
}
```

Caveats

http://www.cs.unm.edu/~angel/WebGL/7E/code_updates.pdf

In Code

http://www.cs.unm.edu/~angel/WebGL/7E/02/gasket4.html

http://www.cs.unm.edu/~angel/WebGL/7E/02/gasket4.js

GLSL Programming Language

Data Types

- C types: int, float, bool
- Vectors:
  - float vec2, vec3, vec4
  - Also int (ivec) and boolean (bvec)
- Matrices: mat2, mat3, mat4
  - Stored by columns
  - Standard referencing m[row][column]
- C++ style constructors
  - vec3 a =vec3(1.0, 2.0, 3.0)
  - vec2 b = vec2(a)

Pointers

- There are no pointers in GLSL
- C structs which can be copied back from functions
- Matrices and vectors can be passed to and fro GLSL functions, e.g. mat3 func(mat3 a)
- Variables passed by copying

Qualifiers

- GLSL has same qualifiers such as **const** as C/C++
- Need others due to nature of execution model

- Variables can change
  - Once per primitive
  - Once per vertex
  - Once per fragment
  - At any time in the application

- Vertex attributes interpolated by rasterizer into fragment attributes

Passing values

- Call by value-return
- Variables are copied in
- Returned values are copied back

Attribute Qualifier

- Attribute-qualified variables change at most once per vertex
- Built-in variables, gl_Position, most deprecated
- User defined (in application program)
  - **attribute float temperature**
  - **attribute vec3 velocity**
  - recent versions of GLSL
    - **in** and **out** qualifiers to get to and from shaders

Uniform Qualified

- Variables constant for an entire primitive
- Changed in application and sent to shaders
- Not changed in shader
- Pass information to shader, time or bounding box, of a primitive or transformation matrices

Varying Qualified

- Variables passed from vertex shader to fragment shader

- Automatically interpolated by the rasterizer
- With WebGL, GLSL uses varying qualifier in both shaders

  varying vec4 color;

- More recent versions of WebGL use out in vertex shader and in in the fragment shader

  out vec4 color; //vertex shader

  in vec4 color; // fragment shader

Naming Convention

- Attributes to vertex shader have names beginning with v (v Position, vColor) in both application & shader
  - Note these are different entities with the same name

- Variable variables begin with f (fColor) in both shaders
  - must have same name

- Uniform variables are unadorned and can have the same name in application and shaders

Example: Vertex Shader

```
attribute vec4 vColor;
varying vec4 fColor;
void main()
{
  gl_Position = vPosition;
  fColor = vColor;
}
```

Fragment Shader

```
precision mediump float;


varying vec3 fColor;
void main()
{
  gl_FragColor = fColor;
}
```

Wave Motion – Varying Qualified

```
in vec4 vPosition;
uniform float xs, zs, // frequencies
uniform float h; // height scale
void main()
{
  vec4 t = vPosition;
  t.y = vPosition.y
     + h*sin(time + xs*vPosition.x)
     + h*sin(time + zs*vPosition.z);
  gl_Position = t;
}
```

Another Example- Particles

```
in vec3 vPosition;
uniform mat4 ModelViewProjectionMatrix;
uniform vec3 init_vel;
uniform float g, m, t;
void main()
{ vec3 object_pos;
object_pos.x = vPosition.x + vel.x*t;
object_pos.y = vPosition.y + vel.y*t
       + g/(2.0*m)*t*t;
object_pos.z = vPosition.z + vel.z*t;
gl_Position =
  ModelViewProjectionMatrix*vec4(object_pos,1);
}
```

Particles Fragment Shader

```
/* pass-through fragment shader */

in vec4 color;
void main(void)
{
    gl_FragColor = color;
}
```

Sending a Uniform Variable

```
GLint aParam;
aParam = glGetUniformLocation(myProgObj,
    "angle");
/* angle defined in shader */

/* my_angle set in application */
GLfloat my_angle;
my_angle = 5.0 /* or some other value */

glUniform1f(aParam, my_angle);
```

Sending a Uniform Variable - 2

```
 // in application
vec4 color = vec4(1.0, 0.0, 0.0, 1.0);
colorLoc = gl.getUniformLocation( program, ”color" );
gl.uniform4f( colorLoc, color);

// in fragment shader (similar in vertex shader)

uniform vec4 color;

void main()
{
    gl_FragColor = color;
}
```

Adding Color

- Send color to the shaders as a vertex attribute or as a uniform variable
- Choice depends on frequency of change
- Associate a color with each vertex
- Set up an array of same size as positions
- Send to GPU as a vertex buffer object

Setting Colors

```
typedef  vec3 color3;
color3 base_colors[4] = {color3(1.0, 0.0. 0.0), ….
color3 colors[NumVertices];
vec3 points[NumVertices];

//in loop setting positions

colors[i] = basecolors[color_index]
position[i] = …….
```

Packing Colors in Application

```
var cBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer );
gl.bufferData( gl.ARRAY_BUFFER, flatten(colors),
                                gl.STATIC_DRAW );

var vColor = gl.getAttribLocation( program, "vColor" );
gl.vertexAttribPointer( vColor, 3, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vColor );
```

Operators and Functions

- Standard C functions
  - Trigonometric
  - Arithmetic
  - Normalize, reflect, length
- Overloading of vector and matrix types

```
mat4 a;
vec4 b, c, d;
c = b*a; // a column vector stored as a 1d array
d = a*b; // a row vector stored as a 1d array
```

Swizzling and Selection

- Can refer to array elements by element using [] or selection (.) operator with
  - x, y, z, w
  - r, g, b, a
  - s, t, p, q
  - a[2], a.b, a.z, a.p are the same
- Swizzling operator lets us manipulate components

```
vec4 a, b;
a.yz = vec2(1.0, 2.0, 3.0, 4.0);
b = a.yxzw;
```

Looking Closely at Code

Looking Closely

```
gl.useProgram( program );

var bufferId = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
gl.bufferData( gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW );

var vPosition = gl.getAttribLocation( program, "vPosition" );
    gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
    gl.enableVertexAttribArray( vPosition );
```

https://www.opengl.org/sdk/docs/man3/xhtml/

https://www.khronos.org/opengles/sdk/docs/

bufferId = gl.createBuffer();

Create a buffer object using createBuffer()

Looking Closely (2)

```
gl.useProgram( program );

var bufferId = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
gl.bufferData( gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW );

var vPosition = gl.getAttribLocation( program, "vPosition" );
    gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
    gl.enableVertexAttribArray( vPosition );
```

gl.bindBuffer(gl.ARRAY_BUFFER, bufferId)

- bind a named buffer object

void glBindBuffer(GLenum target, GLuint buffer)

Parameters

- Target - Specifies the target to which the buffer object is bound. The symbolic constant must be GL_ARRAY_BUFFER or GL_ELEMENT_ARRAY_BUFFER.
- Buffer - Specifies the name of a buffer object.

glBindBuffer lets you create or use a named buffer object.

Calling glBindBuffer with target set to GL_ARRAY_BUFFER or GL_ELEMENT_ARRAY_BUFFER and buffer set to the name of the new buffer object binds the buffer object name to the target. When a buffer object is bound to a target, the previous binding for that target is automatically broken.

https://www.opengl.org/sdk/docs/man3/xhtml/glBindBuffer.xml

https://www.khronos.org/opengles/sdk/docs/man/xhtml/glBindBuffer.xml

Looking Closely (3)

```
gl.useProgram( program );

var bufferId = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
gl.bufferData( gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW );

var vPosition = gl.getAttribLocation( program, "vPosition" );
    gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
    gl.enableVertexAttribArray( vPosition );
```

gl.bufferData(gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW);

glBufferData — creates and initializes a buffer object's data store

void glBufferData(GLenum target,Glsizeiptr size, const GLvoid * data,Glenum usage);

https://www.opengl.org/sdk/docs/man3/xhtml/glBufferData.xml

glBufferData Parameters

Target - Specifies the target buffer object. The symbolic constant must be GL_ARRAY_BUFFER, GL_COPY_READ_BUFFER, GL_COPY_WRITE_BUFFER, GL_ELEMENT_ARRAY_BUFFER, GL_PIXEL_PACK_BUFFER, GL_PIXEL_UNPACK_BUFFER, GL_TEXTURE_BUFFER, GL_TRANSFORM_FEEDBACK_BUFFER, or GL_UNIFORM_BUFFER.

Size - Specifies the size in bytes of the buffer object's new data store.

Data - Specifies a pointer to data that will be copied into the data store for initialization, or NULL if no data is to be copied.

Usage -Specifies the expected usage pattern of the data store. The symbolic constant must be GL_STREAM_DRAW, GL_STREAM_READ, GL_STREAM_COPY, GL_STATIC_DRAW, GL_STATIC_READ, GL_STATIC_COPY, GL_DYNAMIC_DRAW, GL_DYNAMIC_READ, or GL_DYNAMIC_COPY.

Looking Closely(4)

```
gl.useProgram( program );

var bufferId = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
gl.bufferData( gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW );

var vPosition = gl.getAttribLocation( program, "vPosition" );
    gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
    gl.enableVertexAttribArray( vPosition );
```

gl.getAttribLocation(program,"vPosition");

- Returns location of an attribute variable
- gl.getAttribLocation(Gluint program,const Glchar *name);
- Parameters
    - Program – specifies the program object to be queried
    - Name – null terminated string containing the name of attribute variable whose location is to be queried

Description - glGetAttribLocation queries the previously linked program object specified by program for the attribute variable specified by name and returns the index of the generic vertex attribute that is bound to that attribute variable. If name is a matrix attribute variable, the index of the first column of the matrix is returned. If the named attribute variable is not an active attribute in the specified program object or if name starts with the reserved prefix "gl_", a value of -1 is returned.

https://www.khronos.org/opengles/sdk/docs/man/xhtml/glGetAttribLocation.xml

Looking Closely(5)

```
gl.useProgram( program );

var bufferId = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
gl.bufferData( gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW );

var vPosition = gl.getAttribLocation( program, "vPosition" );
    gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
    gl.enableVertexAttribArray( vPosition );
```

gl.vertexAttribPointer(vPosition,2,gl.FLOAT, false, 0,0)

- define an array of generic vertex attribute data

void glVertexAttribPointer(GLuint index, GLint size,GLenum type,GLboolean normalized,GLsizei stride, const GLvoid * pointer);

Parameters

Index - Specifies the index of the generic vertex attribute to be modified.

Size - Specifies the number of components per generic vertex attribute. Must be 1, 2, 3, or 4. The initial value is 4.

Type - Specifies the data type of each component in the array. Symbolic constants GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_FIXED, or GL_FLOAT are accepted. The initial value is GL_FLOAT.

Normalized - Specifies whether fixed-point data values should be normalized (GL_TRUE) or converted directly as fixed-point values (GL_FALSE) when they are accessed.

Stride - Specifies the byte offset between consecutive generic vertex attributes. If stride is 0, the generic vertex attributes are understood to be tightly packed in the array. The initial value is 0.

Pointer - Specifies a pointer to the first component of the first generic vertex attribute in the arra
The initial value is 0.

glVertexAttribPointer (2)

glVertexAttribPointer specifies the location and data format of the array of generic vertex attributes at index index to use when rendering. size specifies the number of components per attribute and must be 1, 2, 3, or 4. type specifies the data type of each component, and stride specifies the byte stride from one attribute to the next, allowing vertices and attributes to be packed into a single array or stored in separate arrays. If set to GL_TRUE, normalized indicates that values stored in an integer format are to be mapped to the range [-1,1] (for signed values) or [0,1] (for unsigned values) when they are accessed and converted to floating point. Otherwise, values will be converted to floats directly without normalization.

Looking Closely(5)

```
gl.useProgram( program );

var bufferId = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, bufferId );
gl.bufferData( gl.ARRAY_BUFFER, flatten(points), gl.STATIC_DRAW );

var vPosition = gl.getAttribLocation( program, "vPosition" );
    gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
    gl.enableVertexAttribArray( vPosition );
```

gl.enableVertexAttribArray(vPosition);

- Enable or disable a generic vertex attribute array

    void glEnableVertexAttribArray(GLuint index);

    void glDisableVertexAttribArray(GLuint index);

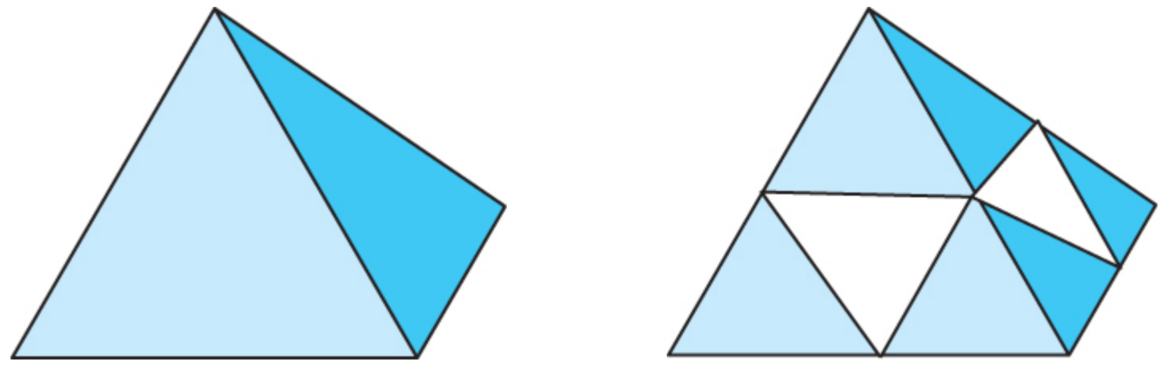
Parameters

Index - Specifies the index of the generic vertex attribute to be enabled or disabled.

Description - glEnableVertexAttribArray enables the generic vertex attribute array specified by index. glDisableVertexAttribArray disables the generic vertex attribute array specified by index. By default, all client-side capabilities are disabled, including all generic vertex attribute arrays.

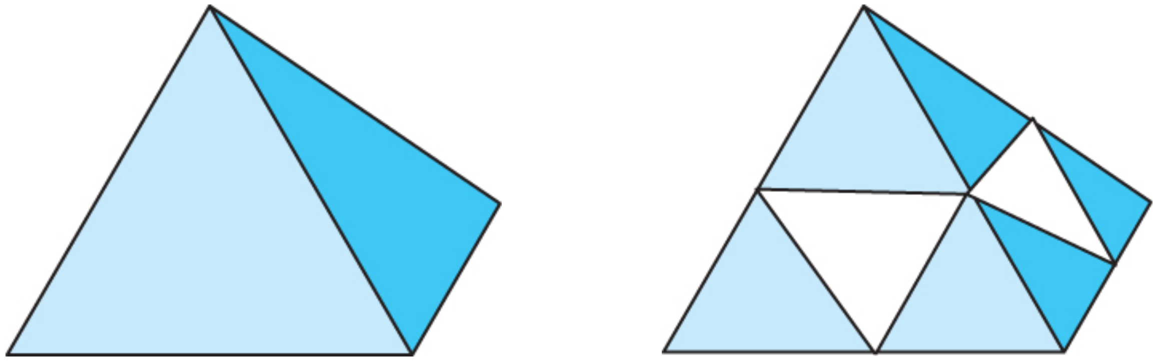
https://www.khronos.org/opengles/sdk/docs/man/xhtml/glEnableVertexAttribArray.xml

3D Gasket



3D Gasket

- We can subdivide each of the four faces



- Appears as if we remove a solid tetrahedron from the center leaving four smaller tetrahedra
- Code almost identical to 2D example

Moving to 3D

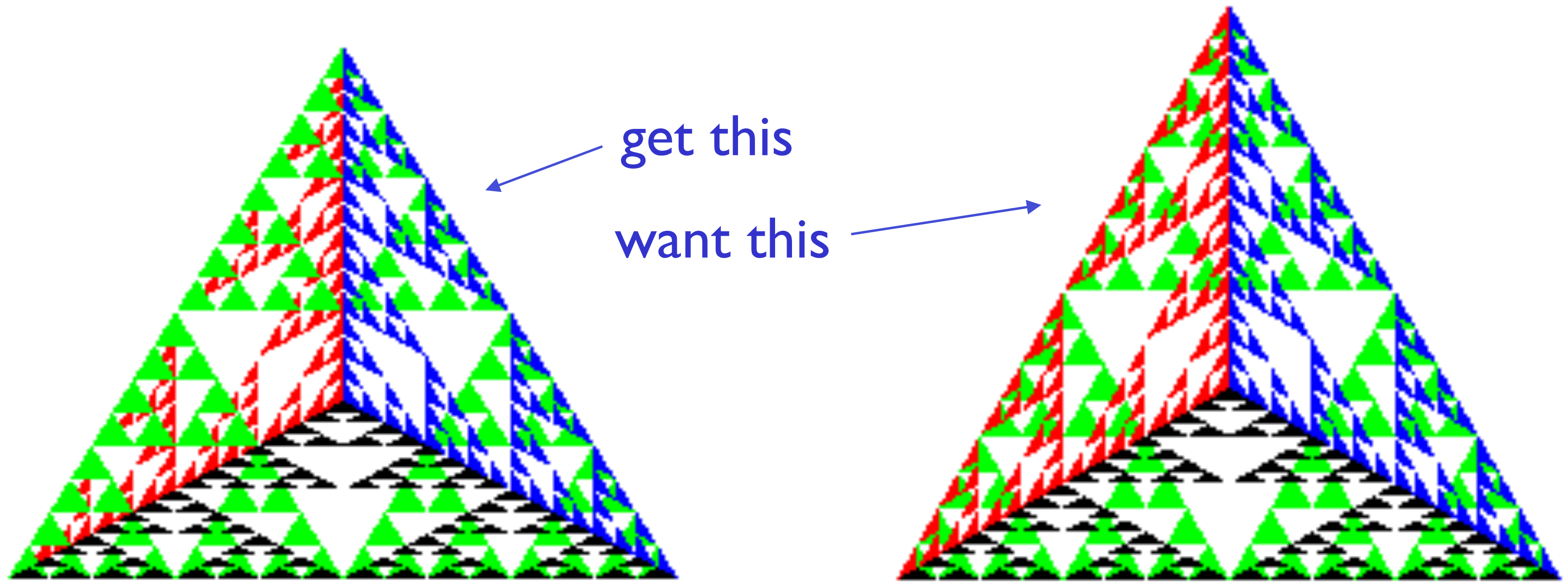
3D points and a tetrahedron

```
var vertices = [
  vec3(  0.0000,  0.0000, -1.0000 ),
   vec3(  0.0000,  0.9428,  0.3333 ),
   vec3( -0.8165, -0.4714,  0.3333 ),
   vec3(  0.8165, -0.4714,  0.3333 )
];
```

subdivide each face

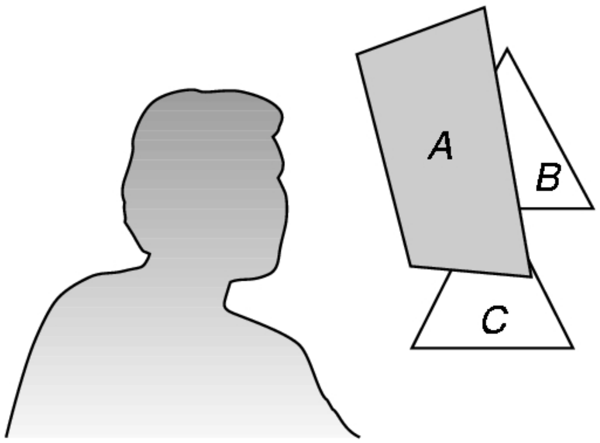
Almost Correct

Because the triangles are drawn in the order they are specified in the program, the front triangles are not always rendered in front of triangles behind them



Hidden-Surface Removal

- We want to see only those surfaces in front of other surfaces
- OpenGL uses a *hidden-surface* method called the *z*-buffer algorithm that saves depth information as objects are rendered so that only the front objects appear in the image



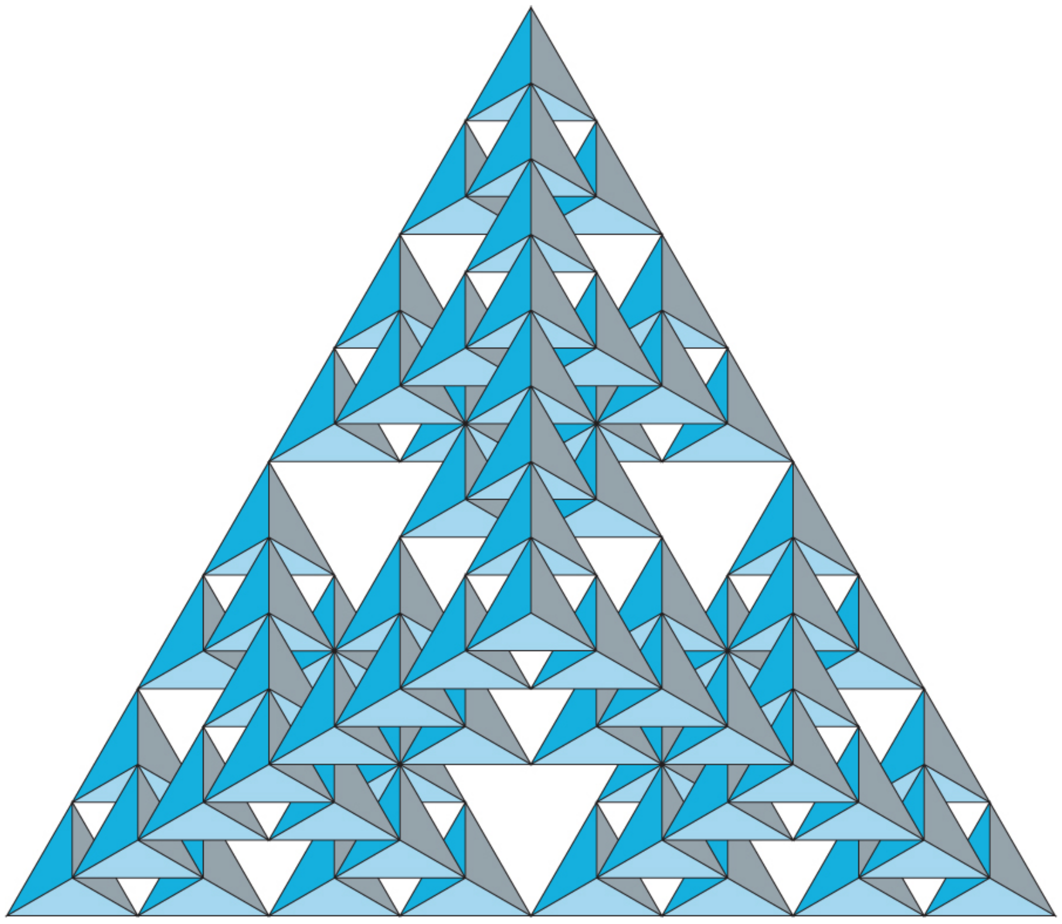
z-buffer Algorithm

- The algorithm uses an extra buffer, the z-buffer, to store depth information as geometry travels down the pipeline
- Depth buffer is required to be available in WebGL
- It must be
  - Enabled
    - gl.enable(gl.DEPTH_TEST)
  - Cleared in for each render
    - gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT)

Surface vs Volume Subdivision

- Previous Example – divided surface of each face
- Divide volume using same midpoints
- Midpoints four smaller tetrahedrons for each vertex
- Smaller tetrahedrons removes a *volume* in the middle

Volume Subdivision



GLSL Shaders - Loading

Linking Shaders w/ Application

- Read shaders
- Compile shaders
- Create a program object
- Link everything together
- Link variables in application with variables in shaders
  - Vertex attributes
  - Uniform variables

Program Object

Container for shaders

- Can contain multiple shaders
- Other GLSL functions

```
var program = gl.createProgram();

gl.attachShader( program, vertShdr );
gl.attachShader( program, fragShdr );
gl.linkProgram( program );
```

Our fav program

```
var program = initShaders( gl, "vertex-shader",
   "fragment-shader" );
 gl.useProgram( program );
```

Reading a Shader

- Shaders are added to the program object and compiled
- Usual method of passing a shader is as a null-terminated string using the function
- gl.shaderSource(fragShdr, fragElem.text);
- If shader is in HTML file, we can get it into application by getElementById method
- If the shader is in a file, we can write a reader to convert the file to a string

HTML example

```
function initShaders( gl, vertexShaderId, fragmentShaderId )
{ var vertShdr; var fragShdr;
    var vertElem = document.getElementById( vertexShaderId );
    if ( !vertElem ) {
        alert( "Unable to load vertex shader " + vertexShaderId ); return -1;
    }
    else {
        vertShdr = gl.createShader( gl.VERTEX_SHADER );
        gl.shaderSource( vertShdr, vertElem.text );
        gl.compileShader( vertShdr );
        if ( !gl.getShaderParameter(vertShdr, gl.COMPILE_STATUS) ) {
            var msg = "Vertex shader failed to compile.  The error log is:"
                + "<pre>" + gl.getShaderInfoLog( vertShdr ) + "</pre>";
            alert( msg );
            return -1;
        }
    }
```

Adding a Vertex Shader

```
var vertShdr;
var vertElem =
    document.getElementById( vertexShaderId );

vertShdr = gl.createShader( gl.VERTEX_SHADER );

gl.shaderSource( vertShdr, vertElem.text );
gl.compileShader( vertShdr );

// after program object created
gl.attachShader( program, vertShdr );
```

Shader Reader

Following code may be a security issue with some browsers if you try to run it locally

- Cross Origin Request

```
function getShader(gl, shaderName, type) {
        var shader = gl.createShader(type);
        shaderScript = loadFileAJAX(shaderName);
        if (!shaderScript) {
            alert("Could not find shader source:
                    "+shaderName);
        }
}
```

Precision Declaration

- In GLSL for WebGL we must specify desired precision in fragment shaders
  - artifact inherited from OpenGL ES
  - ES must run on very simple embedded devices that may not support 32-bit floating point
  - All implementations must support mediump
  - No default for float in fragment shader
- Can use preprocessor directives (#ifdef) to check if highp supported and, if not, default to mediump

Pass Through Fragment Shader

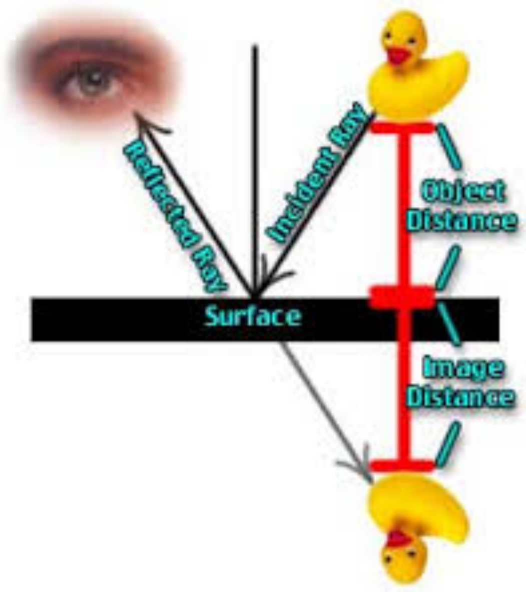
```
#ifdef GL_FRAGMENT_SHADER_PRECISION_HIGH
  precision highp float;
#else
  precision mediump float;
#endif

varying vec4 fcolor;
void main(void)
{
     gl_FragColor = fcolor;
}
```

Next Topic – Linear Algebra

Vectors

- Physical definition:
  - Direction
  - Magnitude
- Examples
  - Light Direction
  - View Direction
  - Normal

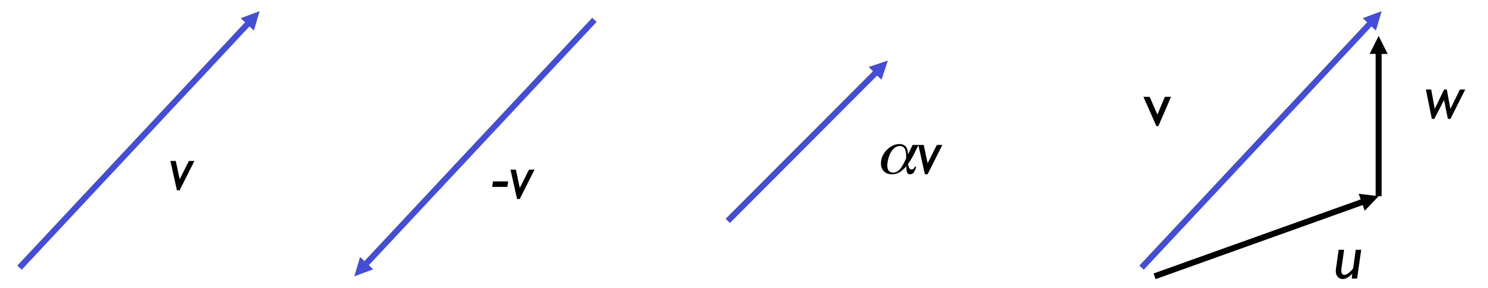


Abstract Spaces

- Scalars
- (Linear) Vector Space
  - Scalars and vectors
- Affine Space
  - Scalars, vectors, and points
- Euclidean Space
  - Scalars, vectors, points
  - Concept of distance
- Projections

Vectors – Linear Space

- Every vector
  - has an inverse
  - can be multiplied by a scalar
- There exists a zero vector
  - Zero magnitude, undefined orientation
- The sum of any two vectors is a vector - closure



Vector Spaces

- Vectors = *n*-tuples $v = (v_1, v_2, \ldots, v_n)$
  - Vector-vector addition

$$\begin{aligned} u + v &= (u_1, u_2, \ldots, u_n) + (v_1, v_2, \ldots, v_n) \\ &= (u_1 + v_1, u_2 + v_2, \ldots, u_n + v_n) \end{aligned}$$

  - Scalar-vector multiplication

$$\alpha v = (\alpha v_1, \alpha v_2, \ldots, \alpha v_n)$$

  - Vector space:

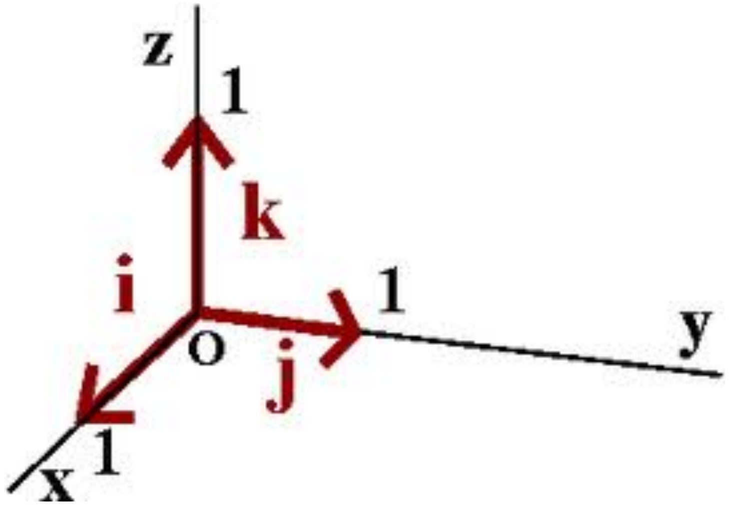
$$\mathbf{R}^n$$

$$\vec{u} = \alpha_1 \vec{u}_1 + \alpha_2 \vec{u}_2 + \cdots + \alpha_n \vec{u}_n$$

Linear Independence

$$\alpha_1\vec{u}_1 + \alpha_2\vec{u}_2 + \cdots + \alpha_n\vec{u}_n = 0$$ iff

$$\alpha_1 = \alpha_2 = \cdots = \alpha_n = 0$$



$$p = (x, y, z) = x\vec{i} + y\vec{j} + z\vec{k}$$

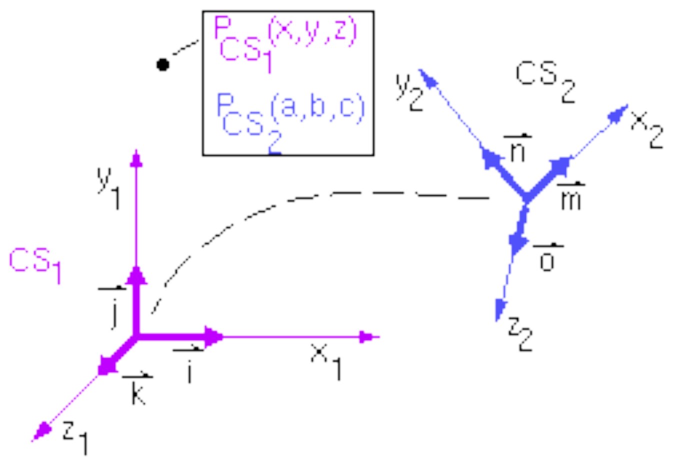
Vector Spaces

- *Dimension*
  - The greatest number of linearly independent vectors
- *Basis* $\{\beta_i\}$
  - *n* linearly independent vectors (*n*: dimension)
- Representation $\vec{v} = \beta_1\vec{v}_1 + \beta_2\vec{v}_2 + \cdots + \beta_n\vec{v}_n$
  - Unique expression in terms of the basis vectors
- <u>Change of Basis</u>: Matrix M
  - Other basis $\vec{v}'_1, \vec{v}'_2, \ldots, \vec{v}'_n$

$$\vec{v} = \beta'_1\vec{v}'_1 + \beta'_2\vec{v}'_2 + \cdots + \beta'_n\vec{v}'_n$$

$$\begin{bmatrix} \beta'_1 \\ \beta'_2 \\ \vdots \\ \beta'_n \end{bmatrix} = \mathbf{M} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}$$

Matrices - Change of Basis



$$\begin{bmatrix} \beta'_1 \\ \beta'_2 \\ \vdots \\ \beta'_n \end{bmatrix} = M \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}$$

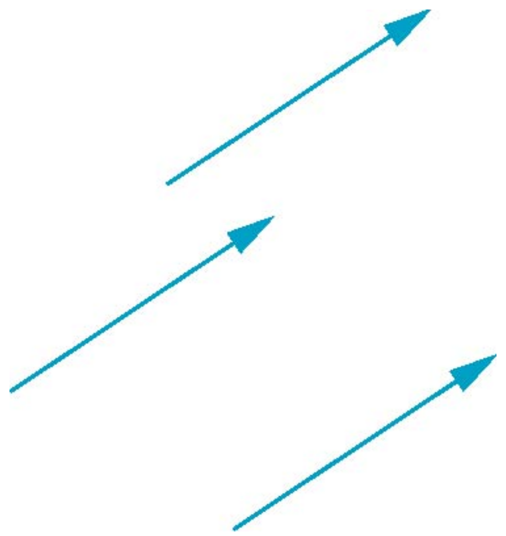
- Change of Basis: Matrix M
  - Other basis

$$\vec{v}'_1, \vec{v}'_2, \ldots, \vec{v}'_n$$

$$\vec{v} = \beta'_1 \vec{v}'_1 + \beta'_2 \vec{v}'_2 + \cdots + \beta'_n \vec{v}'_n$$

Vectors

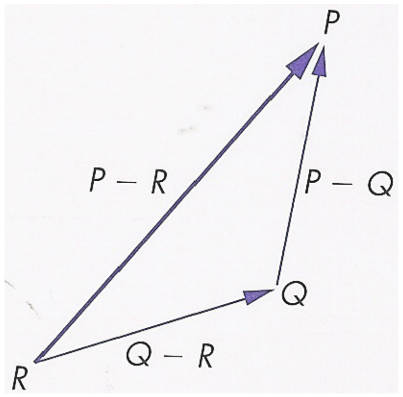
- These vectors are identical
  - Same length and magnitude



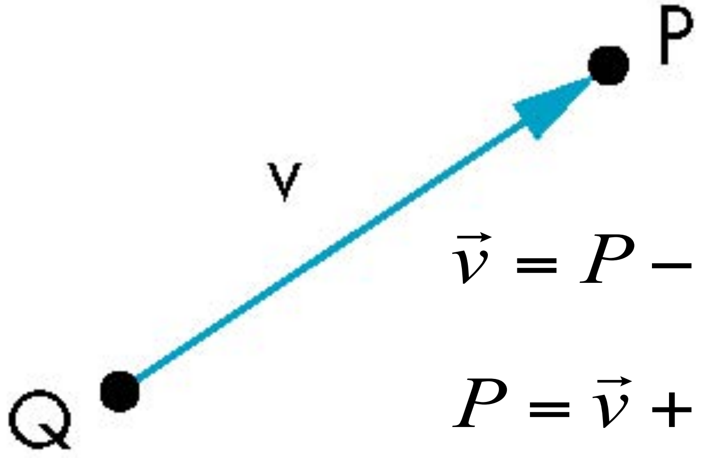
- Vectors spaces insufficient for geometry
  - Need points

Points

- Location in space
- Operations allowed between points and vectors
  - Point-point subtraction yields a vector
  - Equivalent to point-vector addition



$$(P-Q)+(Q-R)=(P-R)$$



$$\vec{v} = P - Q$$

$$P = \vec{v} + Q$$

Affine Spaces

Frame: a Point P_0 and a Set of Vectors $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n$

Representations of the vector and point: *n* scalars

Vector $$v = \alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_n v_n$$

Point $$P = P_0 + \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_n v_n$$

Affine Spaces

- Point + a vector space
- Operations
  - Vector-vector addition
  - Scalar-vector multiplication
  - Point-vector addition
  - Scalar-scalar operations
- For any point define
  - 1 • P = P
  - 0 • P = **0** (zero vector)

Question

How Far Apart Are Two Points in Affine Spaces ?

Operation: ***Inner (dot) Product***

Euclidean (Metric) Spaces

– ***Magnitude*** (***length***) of a vector

$$|v| = \sqrt{v \cdot v}$$

– ***Distance*** between two points

$$|P - Q| = \sqrt{(P - Q) \cdot (P - Q)}$$

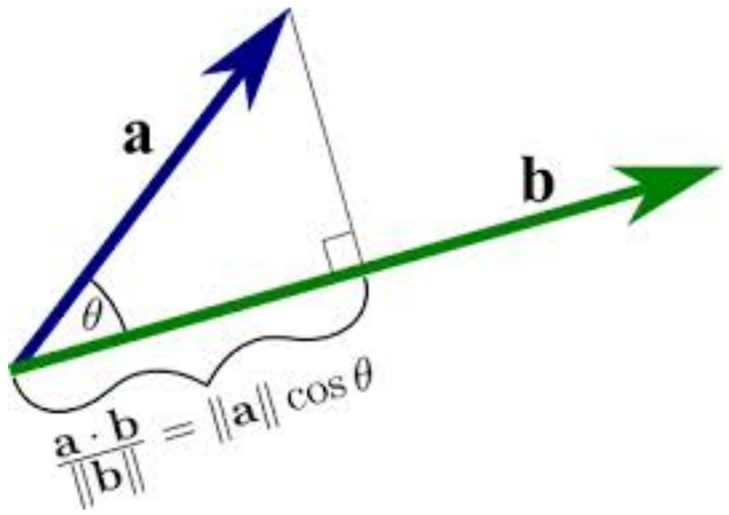
– Measure of the angle between two vectors

$$u \cdot v = |u||v| \cos\theta$$

- cos θ = 0 ➔ orthogonal
- cos θ = 1 ➔ parallel

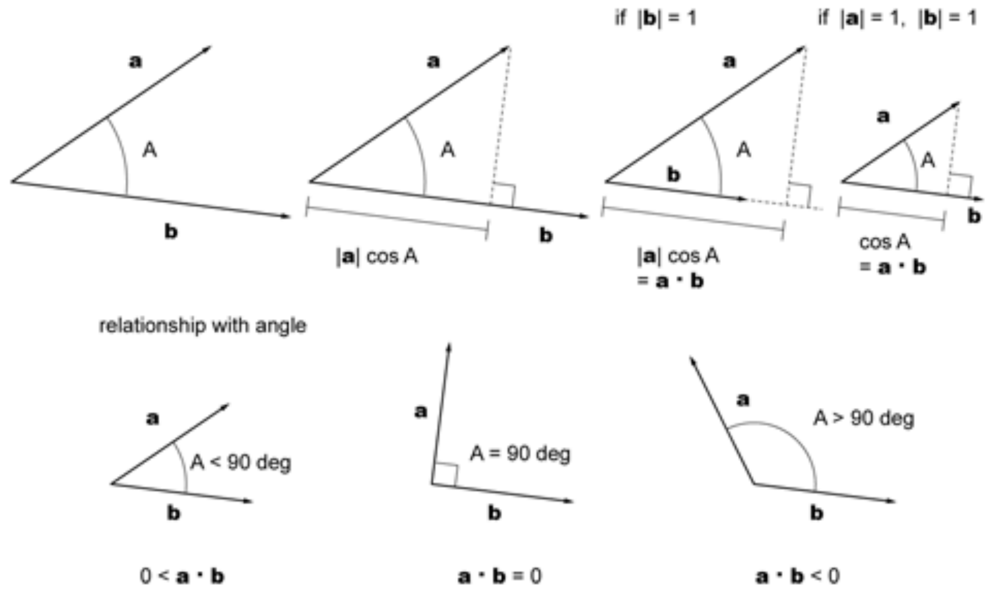
In Pictures

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \bullet \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = a_x.b_x + a_y.b_y + a_z.b_z$$



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

Definition	**a** = (a_x, a_y, a_z)
	b = (b_x, b_y, b_z)
Dot Product (Inner Product) (Scalar Product)	**a** · **b** = $a_x \times b_x + a_y \times b_y + a_z \times b_z$
Geometrical Interpretation	**a** · **b** = |**a**| |**b**| cos A



Euclidean Spaces

– Combine two vectors to form a real

– α, β, γ, …: scalars, *u*, *v*, *w*, …:vectors

$$u \cdot v = v \cdot u$$

$$(\alpha u + \beta v) \cdot w = \alpha u \cdot w + \beta v \cdot w$$

$$v \cdot v > 0 \text{ if } v \neq \mathbf{0}$$

$$\mathbf{0} \cdot \mathbf{0} = 0$$

Orthogonal: $u \cdot v = 0$

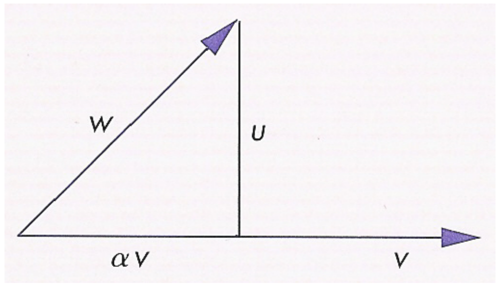
Projections

- Problem: Find shortest distance from a point to a line on a plane
- Given Two Vectors $w = \alpha v + u$
  - Divide into two parts: one parallel and one orthogonal

$$w \cdot v = \alpha v \cdot v + \underline{u} \cdot \underline{v} = \alpha v \cdot v$$

$$\therefore \alpha = \frac{w \cdot v}{v \cdot v}$$

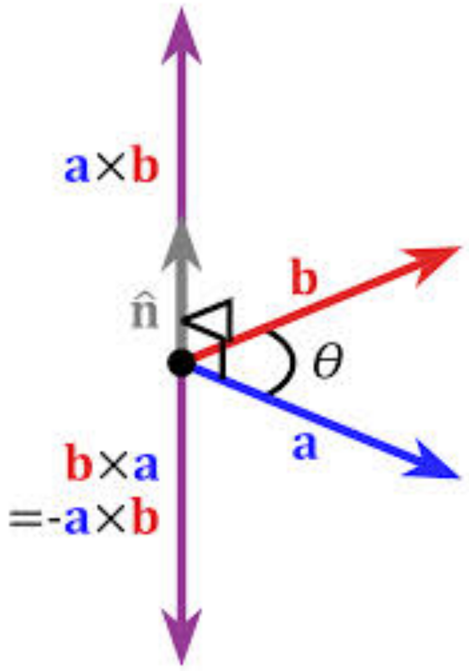
$$\therefore u = w - \alpha v = w - \frac{w \cdot v}{v \cdot v} v$$

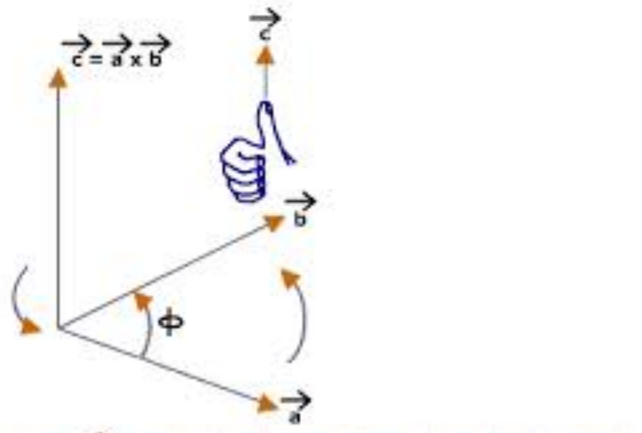


Projection of one vector onto another

Making New Vectors

Cross Product



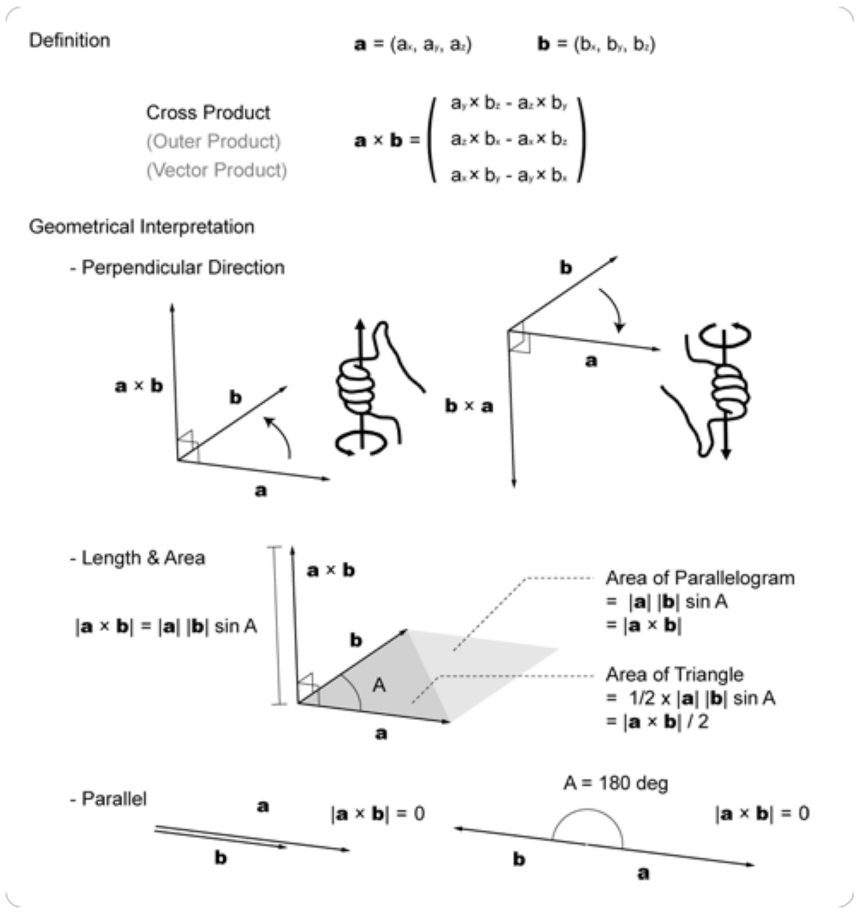


The direction of $\vec{c}$ can also be obtained from the right hand rule

$$\mathbf{c} = \mathbf{a} \times \mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} =$$

$$= (a_2 b_3 - a_3 b_2)\,\mathbf{i} + (a_3 b_1 - a_1 b_3)\,\mathbf{j} + (a_1 b_2 - a_2 b_1)\,\mathbf{k}$$

Cross Product



Definition $\mathbf{a} = (a_x, a_y, a_z)$ $\mathbf{b} = (b_x, b_y, b_z)$

Cross Product (Outer Product) (Vector Product)

$$\mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_y \times b_z - a_z \times b_y \\ a_z \times b_x - a_x \times b_z \\ a_x \times b_y - a_y \times b_x \end{pmatrix}$$

Geometrical Interpretation

- Perpendicular Direction

- Length & Area

$|\mathbf{a} \times \mathbf{b}| = |\mathbf{a}|\,|\mathbf{b}| \sin A$

Area of Parallelogram $= |\mathbf{a}|\,|\mathbf{b}| \sin A = |\mathbf{a} \times \mathbf{b}|$

Area of Triangle $= 1/2 \times |\mathbf{a}|\,|\mathbf{b}| \sin A = |\mathbf{a} \times \mathbf{b}| / 2$

- Parallel

$|\mathbf{a} \times \mathbf{b}| = 0$

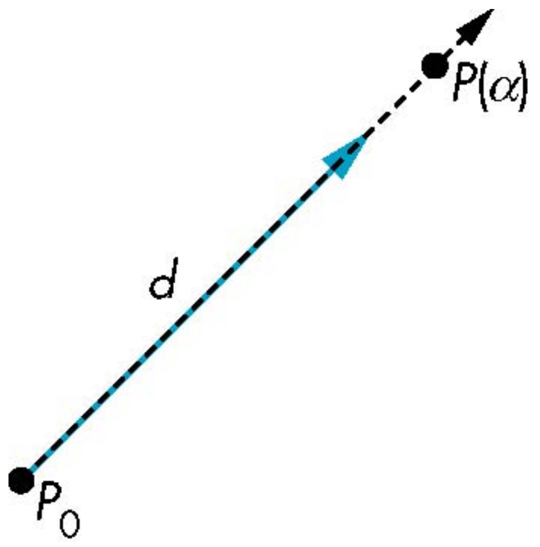
A = 180 deg

$|\mathbf{a} \times \mathbf{b}| = 0$

Parametric Forms

Lines, Rays

- Consider all points of the form
  - $P(\alpha)=P_0 + \alpha\, \mathbf{d}$
  - Set of all points that pass through P_0 in the direction of the vector **d**



2D Forms for lines

- Two-dimensional forms
  - Explicit: $y = mx + h$
  - Implicit: $ax + by + c = 0$
  - Parametric:

$$x(a) = ax_0 + (1-a)x_1$$
$$y(a) = ay_0 + (1-a)y_1$$

Rays, Line Segments

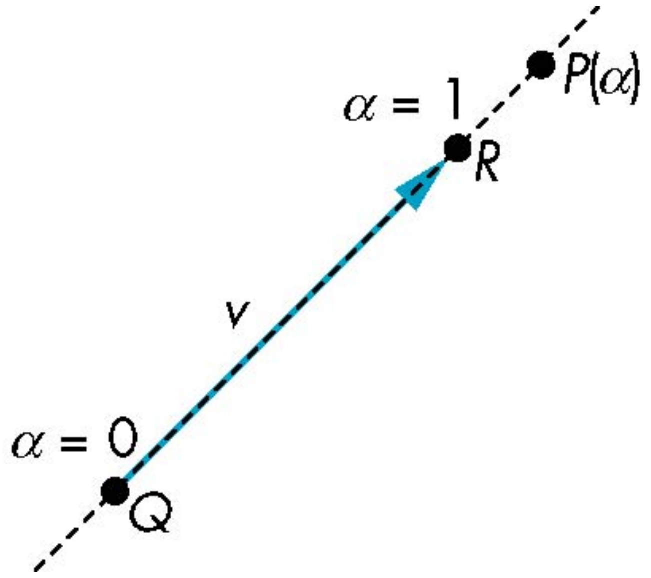
If a >= 0, then P(a) is the *ray* leaving P_0 in the direction **d**

If we use two points to define v, then

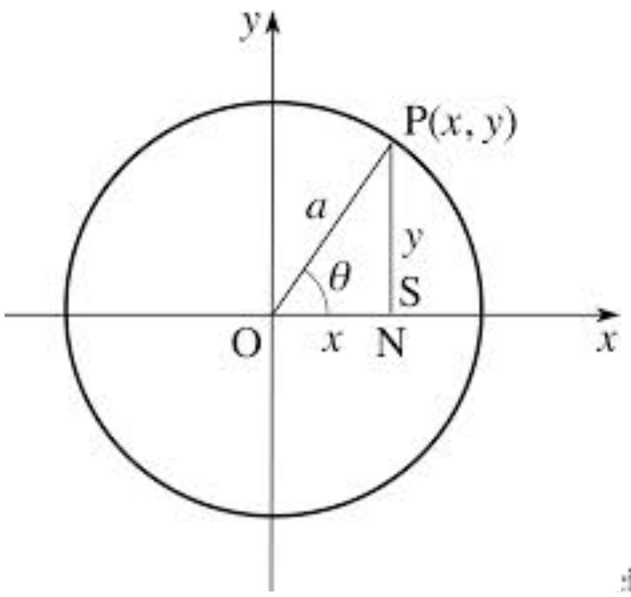
P(a) = Q + a (R-Q)=Q+av

=aR + (1-a)Q

For 0<=a<=1 we get all the points on the *line segment* joining R and Q



Curves

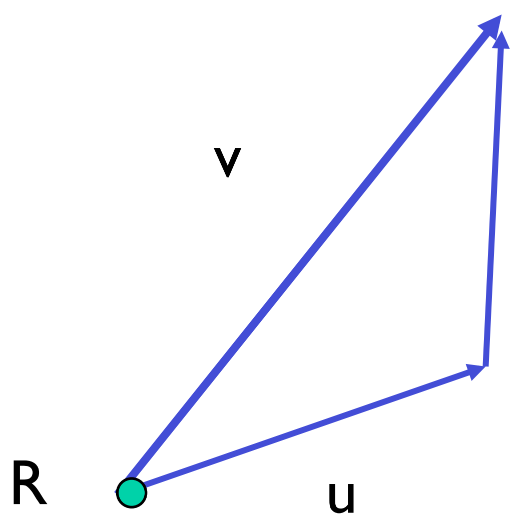


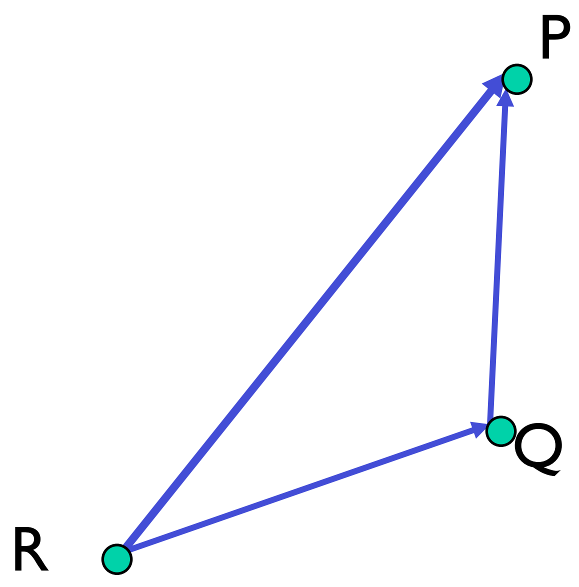
$$\begin{cases} x = R\cos t \\ y = R\sin t \end{cases}, \qquad 0 \le t \le 2\pi,$$

	implicit form	parametric form	
ircle	$x^2 + y^2 - r^2 = 0$	$x(t) = r\frac{1-t^2}{1+t^2}$	$y(t) = r\frac{2t}{1+t^2}$
ellipse	$\frac{x^2}{a^2} + \frac{y^2}{b^2} - 1 = 0$	$x(t) = a\frac{1-t^2}{1+t^2}$	$y(t) = b\frac{2t}{1+t^2}$
hyperbola	$\frac{x^2}{a^2} - \frac{y^2}{b^2} - 1 = 0$	$x(t) = a\frac{1+t^2}{1-t^2}$	$y(t) = b\frac{2t}{1-t^2}$
parabola	$y^2 - 2px = 0$	$x(t) = \frac{t^2}{2p}$	$y(t) = t$

Planes

Defined by a point and two vectors or by three points

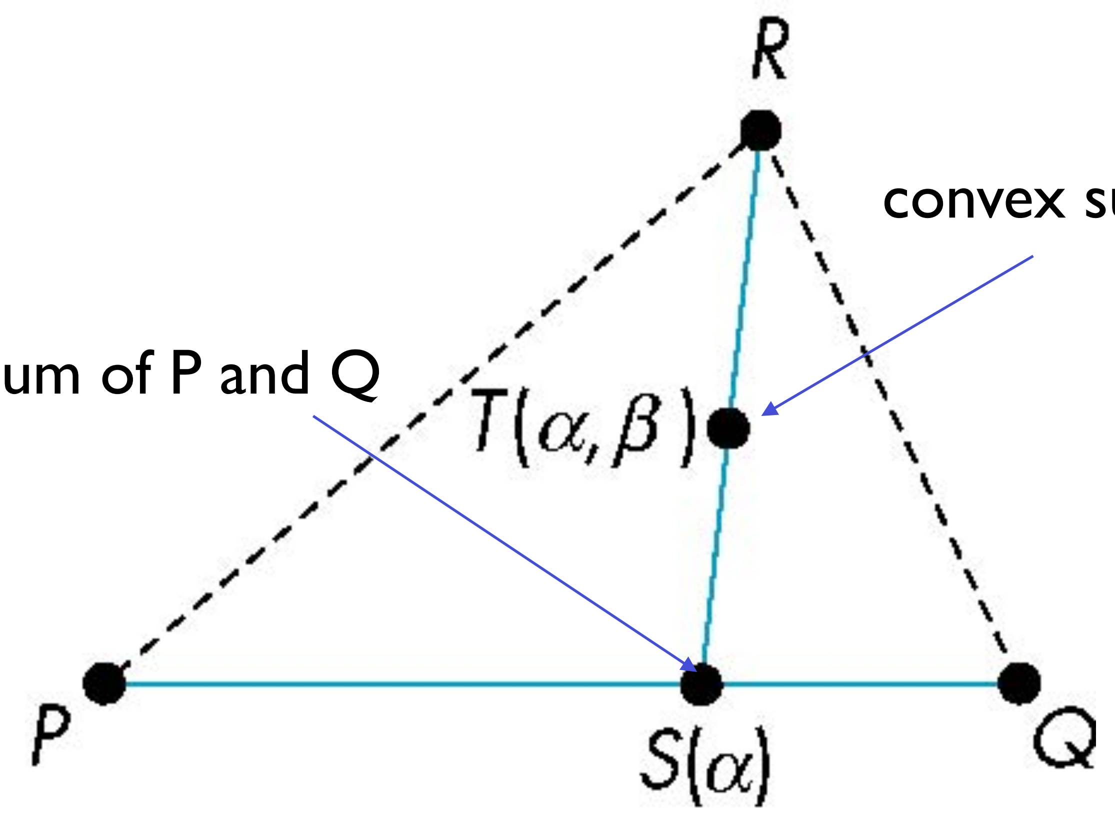




P(a,b)=R+au+bv

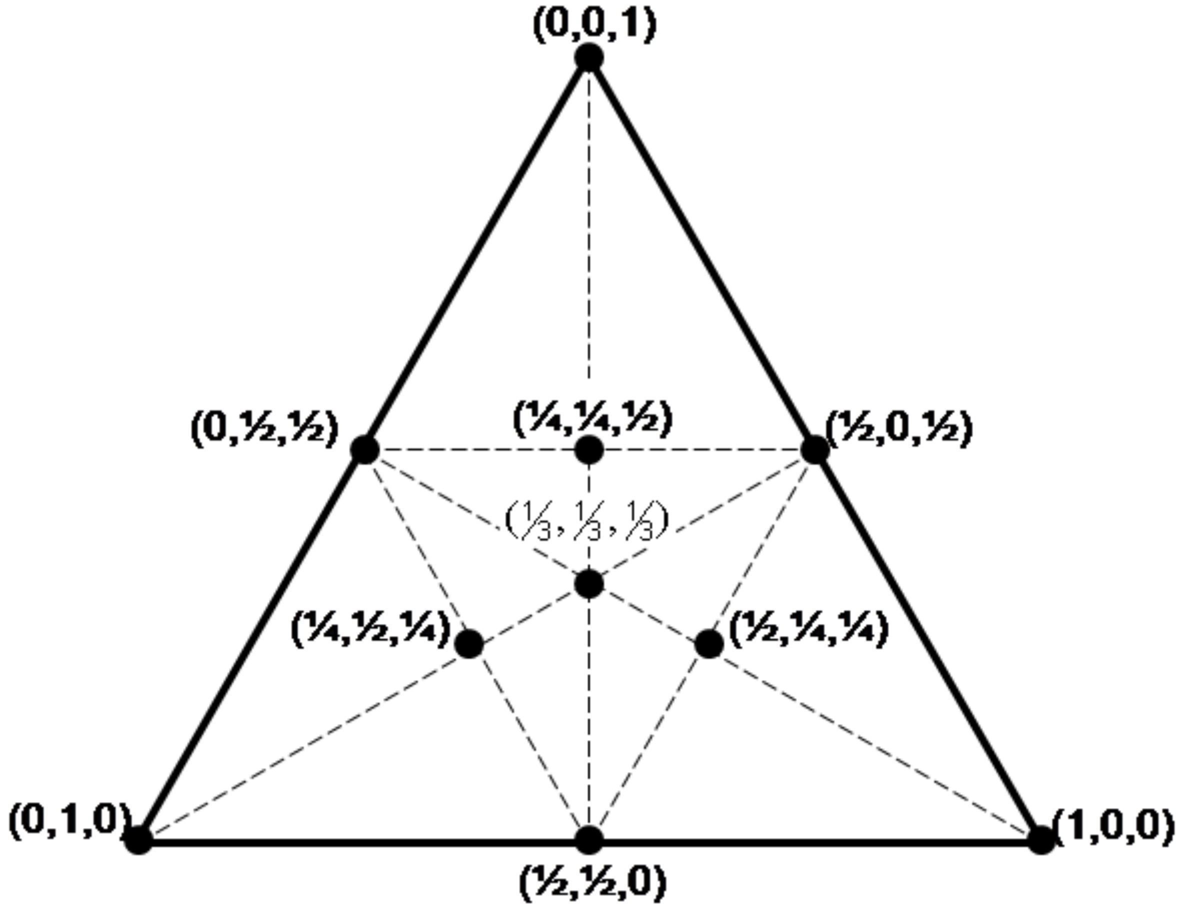
P(a,b)=R+a(Q-R)+b(P-Q)

Triangles



for $0 <= \alpha, \beta <= 1$, we get all points in triangle

Barycentric Coordinates



Barycentric Coordinates

Triangle is convex

Any point inside can be represented as an affine sum

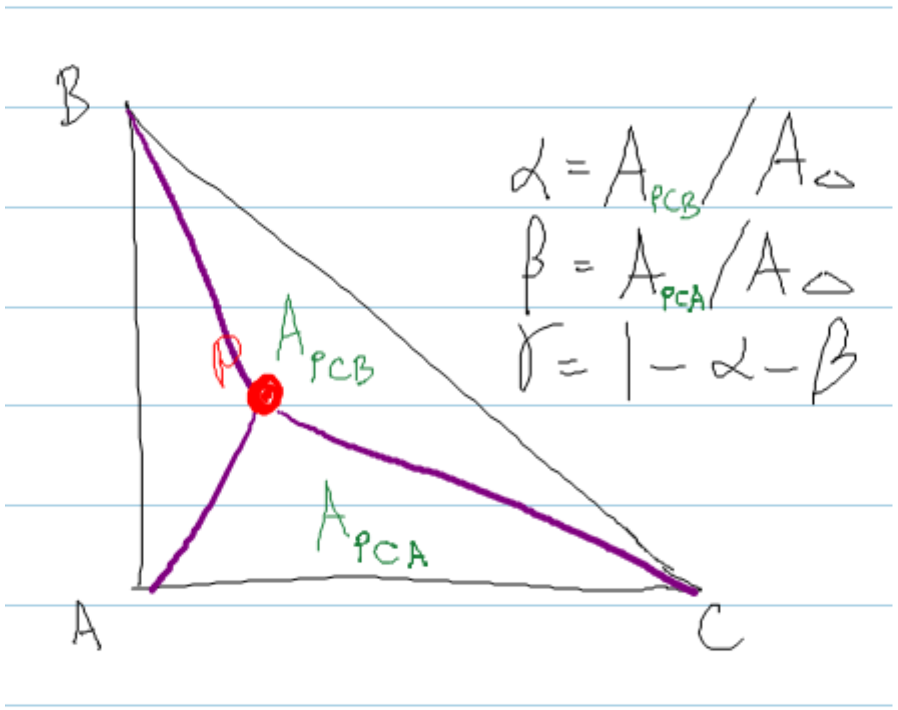
$P(\alpha_{1,}\ \alpha_{2,}\ \alpha_3)=\alpha\, A + \beta B + \gamma C$

where

$\alpha + \beta + \gamma = 1$

$\alpha_{,}\ \beta,\ \gamma >=0$

Barycentric Coordinates


B
A
C
P
A_{PCB}
A_{PCA}
$\alpha = A_{PCB} / A_{\triangle}$
$\beta = A_{PCA} / A_{\triangle}$
$\gamma = 1 - \alpha - \beta$

Calculating Areas ?

Matrices

Matrices

- Definitions
- Matrix Operations
- Row and Column Matrices
- Rank
- Change of Representation
- Cross Product

What is a Matrix?

Elements, organized into rows and columns

$$\begin{array}{cc} & \xrightarrow{\text{rows}} \\ \text{columns}\Big\downarrow & \begin{bmatrix} a & b \\ c & d \end{bmatrix} \end{array}$$

Definitions $\mathbf{A} = \left[a_{ij}\right]$

n x *m* Array of Scalars (*n* Rows and *m* Columns)

- *n*: row *dimension* of a matrix, *m*: column *dimension*
- *m* = *n*: *square matrix* of dimension *n*
- Element

  $\{a_{ij}\}, \ i = 1, \ldots, n, \ \ j = 1, \ldots, m$
- *Transpose*: interchanging the rows and columns of a matrix $\mathbf{A}^T = \left[a_{ji}\right]$

• Column Matrices and Row Matrices

- *Column matrix* (*n* x 1 matrix):
- *Row matrix* (1 x *n* matrix):

$$\mathbf{b} = \left[b_i\right] = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

$$\mathbf{b}^T$$

Basic Operations

Addition, Subtraction, Multiplication

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}$$

add elements

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} - \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a-e & b-f \\ c-g & d-h \end{bmatrix}$$

subtract elements

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

Multiply each row by each column

Matrix Operations

- *Scalar-Matrix Multiplication*

$$\alpha\mathbf{A} = \lfloor \alpha a_{ij} \rfloor$$

- *Matrix-Matrix Addition*

$$\mathbf{C} = \mathbf{A} + \mathbf{B} = \lfloor a_{ij} + b_{ij} \rfloor$$

- *Matrix-Matrix Multiplication*
  - A: *n* x *l* matrix, B: *l* x *m* ➔ C: *n* x *m* matrix

$$\mathbf{C} = \mathbf{AB} = \lfloor c_{ij} \rfloor$$

$$c_{ij} = \sum_{k=1}^{l} a_{ik} b_{kj}$$

Matrix Operations

- Properties of Scalar-Matrix Multiplication

$$\alpha(\beta\mathbf{A}) = (\alpha\beta)\mathbf{A}$$

$$\alpha\beta\mathbf{A} = \beta\alpha\mathbf{A}$$

- Properties of Matrix-Matrix Addition
  - Commutative: $\mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A}$
  - Associative: $\mathbf{A} + (\mathbf{B} + \mathbf{C}) = (\mathbf{A} + \mathbf{B}) + \mathbf{C}$
- Properties of Matrix-Matrix Multiplication

$$\mathbf{A}(\mathbf{BC}) = (\mathbf{AB})\mathbf{C}$$

$$\mathbf{AB} \neq \mathbf{BA}$$

- ***Identity Matrix*** **I** (Square Matrix)

$$\mathbf{I} = [a_{ij}], \quad a_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{AI} = \mathbf{A}$$

$$\mathbf{IB} = \mathbf{B}$$

Identity Matrix

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Multiplication

- Is AB = BA? Maybe, but maybe not!

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & \ldots \\ \ldots & \ldots \end{bmatrix} \qquad \begin{bmatrix} e & f \\ g & h \end{bmatrix}\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} ea+fc & \ldots \\ \ldots & \ldots \end{bmatrix}$$

- Heads up: multiplication is NOT commutative!

Row and Column Matrices

- Column Matrix
  - p^T: row matrix
- *Concatenations*
  - Associative
- By Row Matrix

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{Ap}$$

$$\mathbf{p}' = \mathbf{ABCp}$$

$$(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$$

$$\mathbf{p}'^T = \mathbf{p}^T\mathbf{C}^T\mathbf{B}^T\mathbf{A}^T$$

Inverse of a Matrix

- Identity matrix:
  $AI = A$
- Some matrices have an inverse, such that:
  $AA^{-1} = I$
- Inversion is tricky:
  $(ABC)^{-1} = C^{-1}B^{-1}A^{-1}$
- Derived from non-commutativity property

Determinant of a Matrix

- Used for inversion
- If det(A) = 0, then A has no inverse
- Can be found using factorials, pivots, and cofactors!
- And for Areas of Triangles

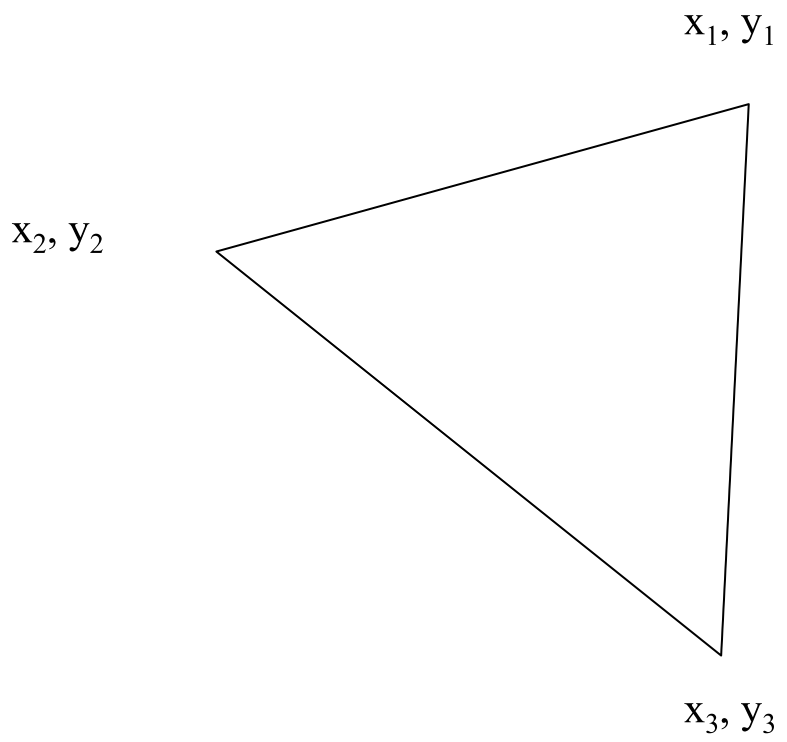
$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\det(A) = ad - bc$$

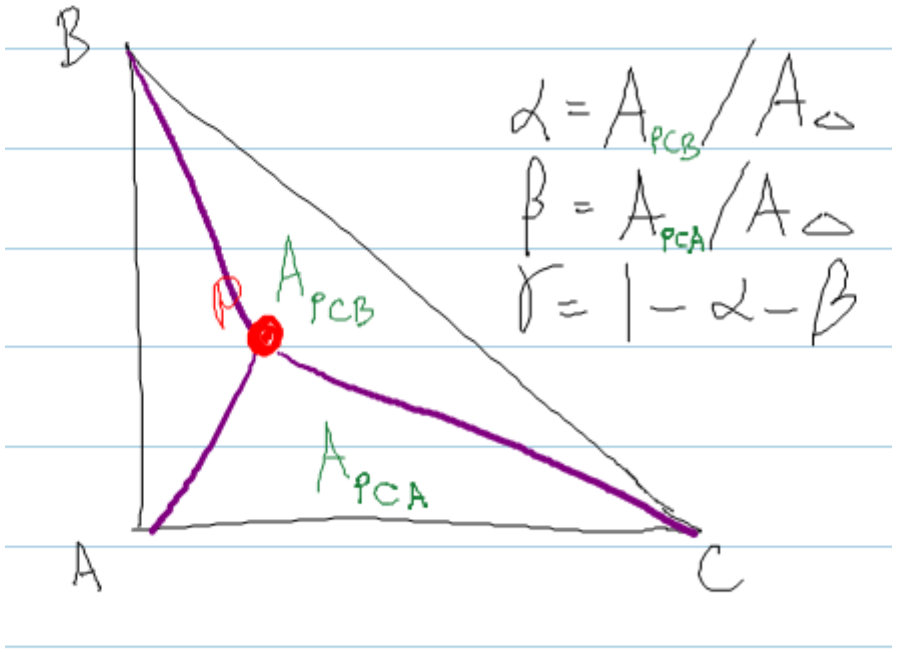
$$A^{-1} = \frac{1}{ad - bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Area of Triangle – Cramer's Rule

$$Ans = \frac{1}{2} \bullet \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$



Use This Here


B
A
C
P
A_{PCB}
A_{PCA}
$\alpha = A_{PCB} / A_\triangle$
$\beta = A_{PCA} / A_\triangle$
$\gamma = 1 - \alpha - \beta$

Change of Representation (1/2)

Change between the Two Bases

$$\{u_1, u_2, \ldots, u_n\} \qquad \{v_1, v_2, \ldots, v_n\}$$

$$v = \alpha_1 u_1 + \alpha_2 u_2 + \cdots + \alpha_n u_n$$

$$v = \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_n v_n$$

– Representations of *v*

$$\mathbf{a} = \begin{bmatrix}\alpha_1 & \alpha_2 & \ldots & \alpha_n\end{bmatrix}^T \qquad \mathbf{b} = \begin{bmatrix}\beta_1 & \beta_2 & \ldots & \beta_n\end{bmatrix}^T$$

– Expression of $\{u_1, u_2, \ldots, u_n\}$ in the basis $\{v_1, v_2, \ldots, v_n\}$

$$u_i = \gamma_{i1} v_1 + \gamma_{i2} v_2 + \cdots + \gamma_{in} v_n, \qquad i = 1, \ldots, n$$

Change of Representation

– $\mathbf{A} = [\gamma_{ij}]$ *n* x *n* matrix

$$\begin{bmatrix} \vec{u}_1 \\ \vec{u}_2 \\ \vdots \\ \vec{u}_n \end{bmatrix} = \mathbf{A} \begin{bmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vdots \\ \vec{v}_n \end{bmatrix}$$

– and $\mathbf{a} = [\alpha_i]$ $\quad \mathbf{b} = [\beta_i]$

$$v = \mathbf{a}^T \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \quad \text{or} \quad v = \mathbf{b}^T \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

– By direct substitution

$$\mathbf{a}^T \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} = \mathbf{b}^T \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \Rightarrow \mathbf{a}^T \mathbf{A} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \mathbf{b}^T \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \Rightarrow \therefore \mathbf{b}^T = \mathbf{a}^T \mathbf{A}$$

Transformations