Graphics Software Development with OpenGL/WebGL

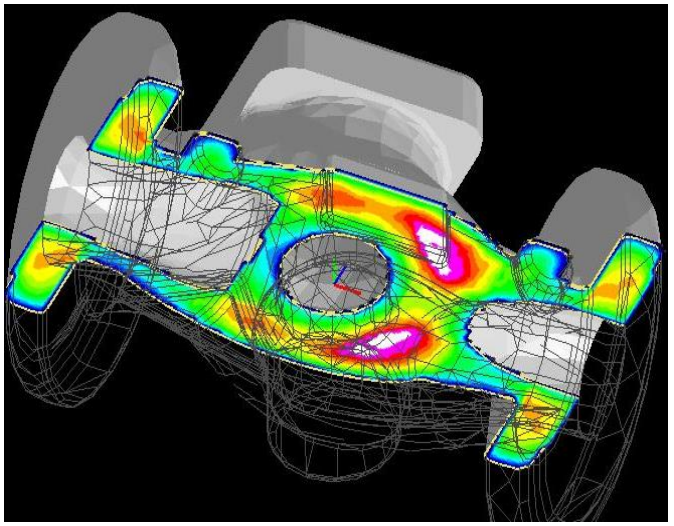
2019.07.08

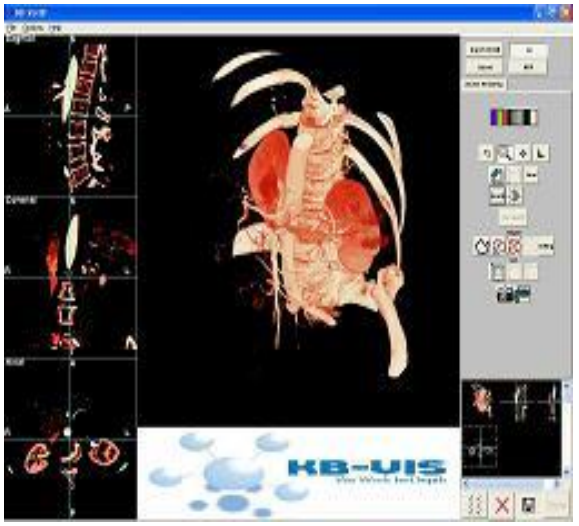
Rapiscan Systems

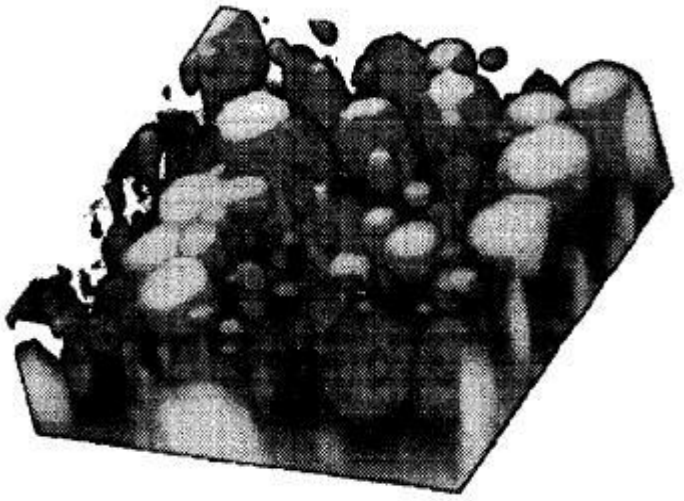
Karthik Bala
KBVIS Technologies Pvt. Ltd.
3d@kbvis.com

About KBVIS

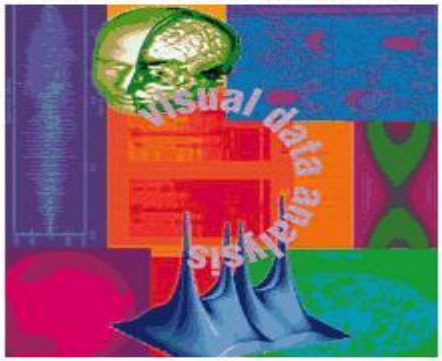
KB-VIS
We Work In-Depth



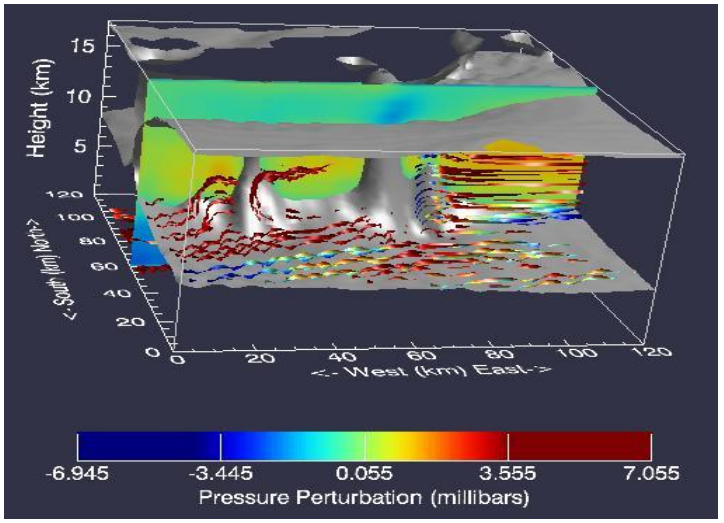


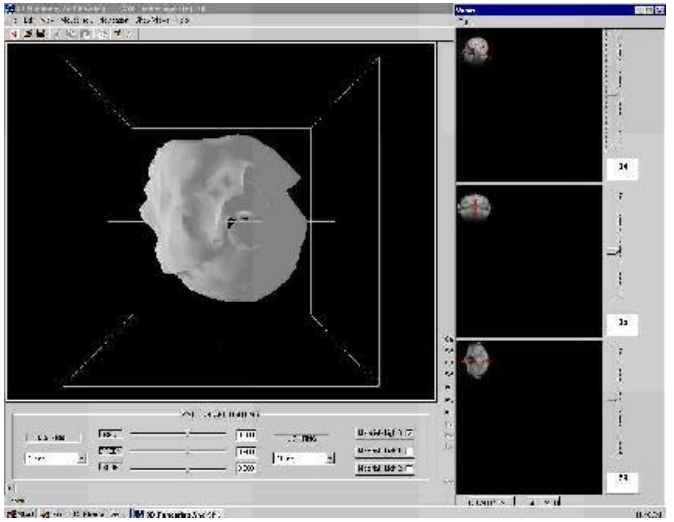


Research Systems, Inc.



IDL®





DAY 1

Setup and Resources

- Presentation Slides (will be updated progressively)

http://kbvis.com/downloads/preso.zip

- WebGL Examples:

http://kbvis.com/downloads/kbvis-examples-webgl.zip

- WebGL Inspector:

http://benvanik.github.io/WebGL-Inspector/

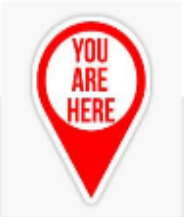
- Latest version of Google Chrome:

https://www.google.com/intl/en/chrome/browser/

- Launch chrome.exe with
- "*--allow-file-access-from-files*" option to load textures and models locally. (As this undermines browser security, do NOT use this mode for browsing or deployment)
- "*--enable-unsafe-es3-apis*" option to enable WebGL 2.0

KB-VIS
We Work In-Depth

Khronos Visual Computing Ecosystem



REAL-TIME 2D / 3D

Cross-platform gaming & UI
VR and AR displays
CAD and product design
Safety-critical displays

Vulkan
OpenGL ES
OpenGL
OpenGL SC
OpenVX

3D FOR THE WEB

3D apps and games in-browser
Runtime delivery of 3D assets

WebGL
glTF
COLLADA

VR, VISION, NEURAL NETWORKS

VR system portability
Tracking and odometry
Scene analysis/understanding
Neural Network inferencing

OpenXR
OpenVX
NNEF

PARALLEL COMPUTATION

Machine learning acceleration
Embedded vision processing
High PerformanceComputing (HPC)

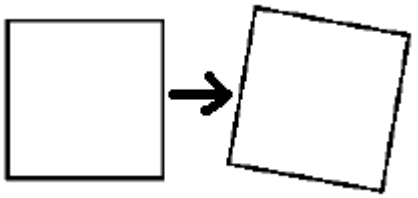
SPIR
SYCL
OpenCL

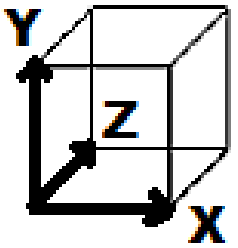
What is OpenGL?

- Software Interface to Graphics Hardware
- OpenGL is a Standard
  - Vendors implement the API (NVIDIA, AMD etc.)
  - Implementations conform to core standard
  - Implementations have own performance characteristics and extensions
- OpenGL is Low-Level
- Toolkits may be built on top of OpenGL

The Big Picture

- To effectively use OpenGL, you need to understand:
  - 3D Graphics Fundamentals:
    - 3D Coordinate Systems
    - Transformations
    - Geometry
    - Lighting & Shading
  - Essential Math:
    - Matrix Algebra
    - Vector Algebra
    - Trigonometry, Linear Algebra, Calculus





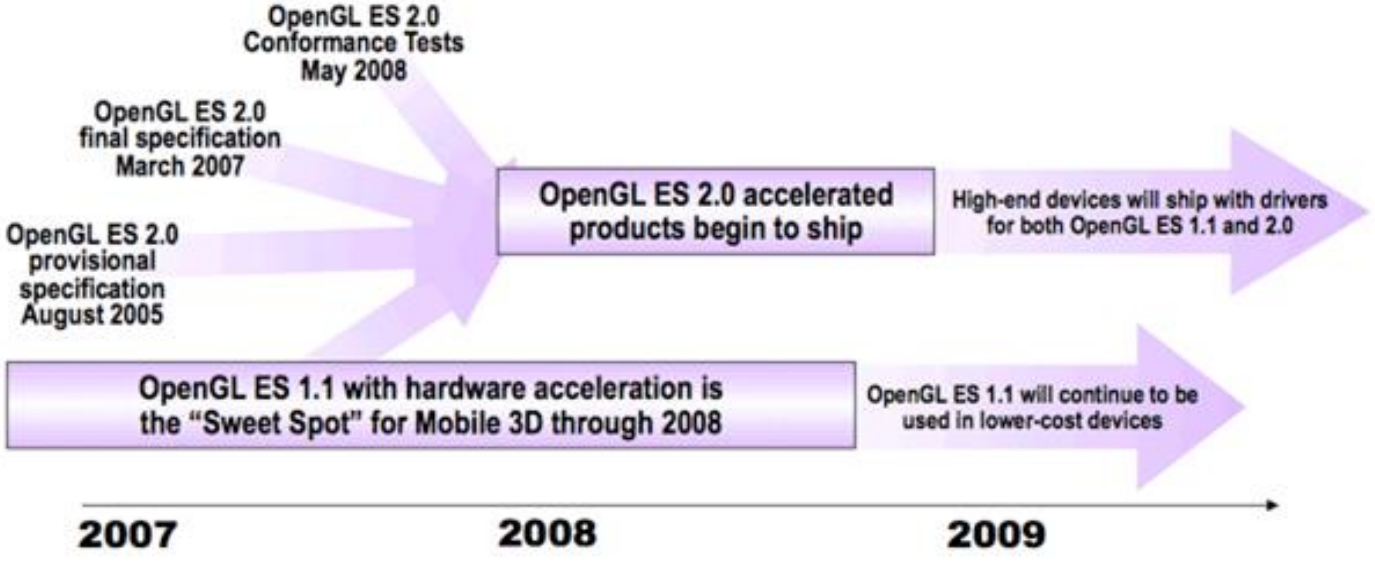
$$\begin{bmatrix} 0 & -z' & y' \\ z' & 0 & -x' \\ -y' & x' & 0 \end{bmatrix}$$

OpenGL Timeline

- Evolved from SGI's IrisGL
- 1992 - Creation of OpenGL Architectural Review Board (ARB) and OpenGL 1.0
- 1997 – 2003 – OpenGL 1.1 – 1.5
- 2003 – OpenGL-ES 1.0
- 2004 – OpenGL 2.0
- 2006 – Control transferred to Khronos group
- 2008 - OpenGL 3.0
- 2010 – 2014 – OpenGL 4.0 - 4.5
- 2011 – WebGL 1.0
- 2017 – WebGL 2.0

OpenGL-ES Overview

- OpenGL for Embedded Systems – phones, consoles, appliances, vehicles
- Lightweight API using well-defined subset profiles of desktop OpenGL
- OpenGL ES 1.0 - for fixed function hardware
- OpenGL ES 2.0 - programmable 3D graphics
- OpenGL ES 3.0 – precision, texture improvements



What's not in OpenGL-ES

- Features that were less popular in modern GL or that map poorly to modern hardware
- Selection, Feedback, Evaluators, Display lists, Occlusion queries
- Color index mode
- Quad, quad strip and general polygon primitives
- Texture proxies, prioritization, and residency
- Vertex-wise rendering (glBegin, glEnd)
- Polygon mode (the mode is always GL_FILL)
- Line and polygon stippling (can be emulated)

What is WebGL?

- API for 3D Graphics on the web
- WebGL 1.0 based on OpenGL-ES 2.0 (2011)
- WebGL 2.0 based on OpenGL-ES 3.0 (2017)
- Implemented in several browsers, including Google Chrome, Mozilla Firefox, Safari, Opera
- Hardware-accelerated 3D graphics inside the browser
- Exposed through HTML5 Canvas element
- Runs natively in the browser - no plug-in needed

Hardware Acceleration

- Hardware-Accelerated Rendering
  - Achieved using a GPU (Graphics Processing Unit)
  - Fast
  - Hardware resources are limited
- Software Implementation
  - e.g. default Windows driver, MESA
  - driver may fall back to software if hardware implementation not available

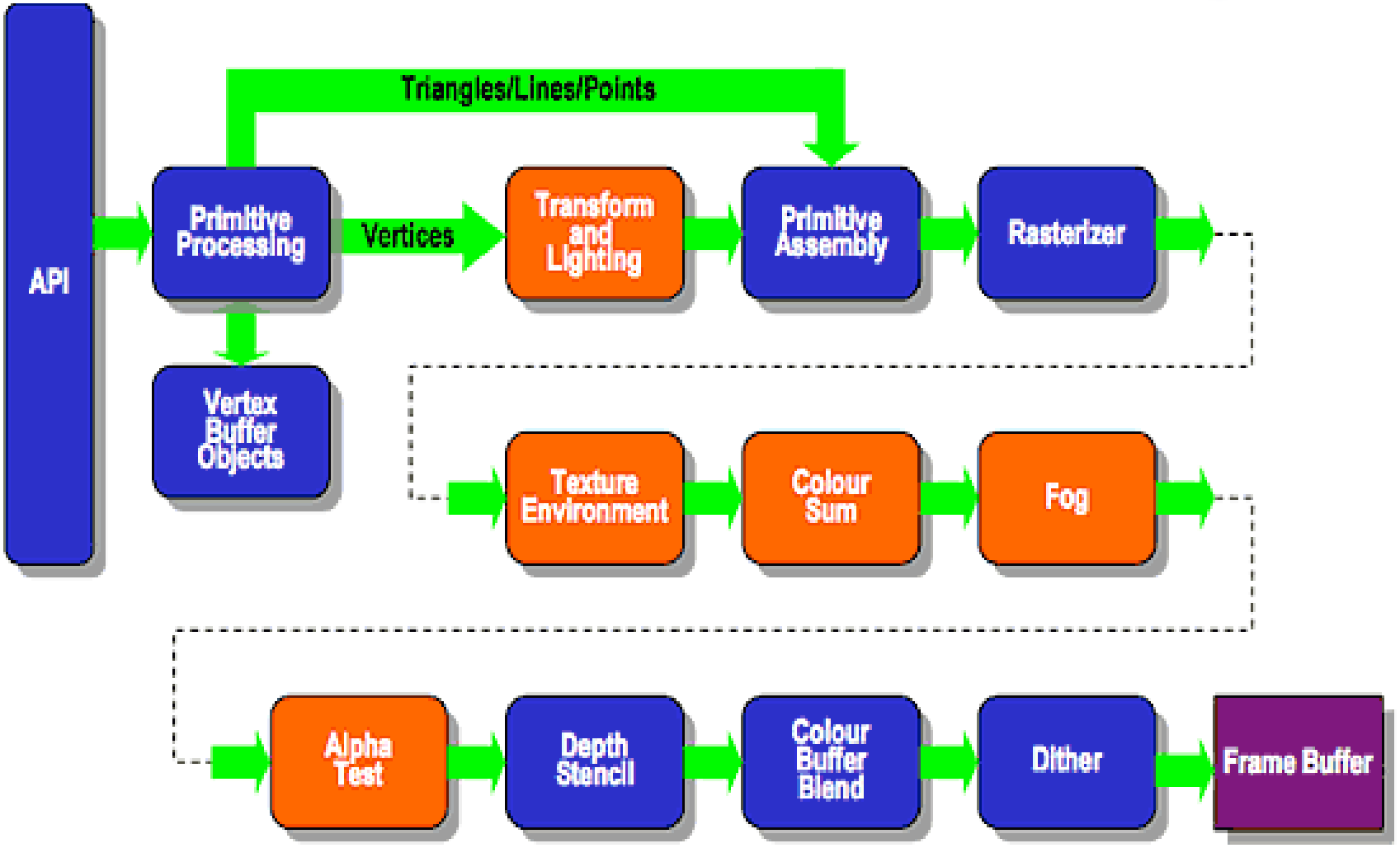
Window System

OpenGL is Platform Independent

- Need windowing system for
  - On-Screen Output
  - Opening/Closing Windows
  - Interaction, Handling events
- Options:
  - GLX (*nix)
  - WGL (windows)
  - EGL (embedded)
  - GLUT (window-system independent)
  - HTML5 Canvas (WebGL)

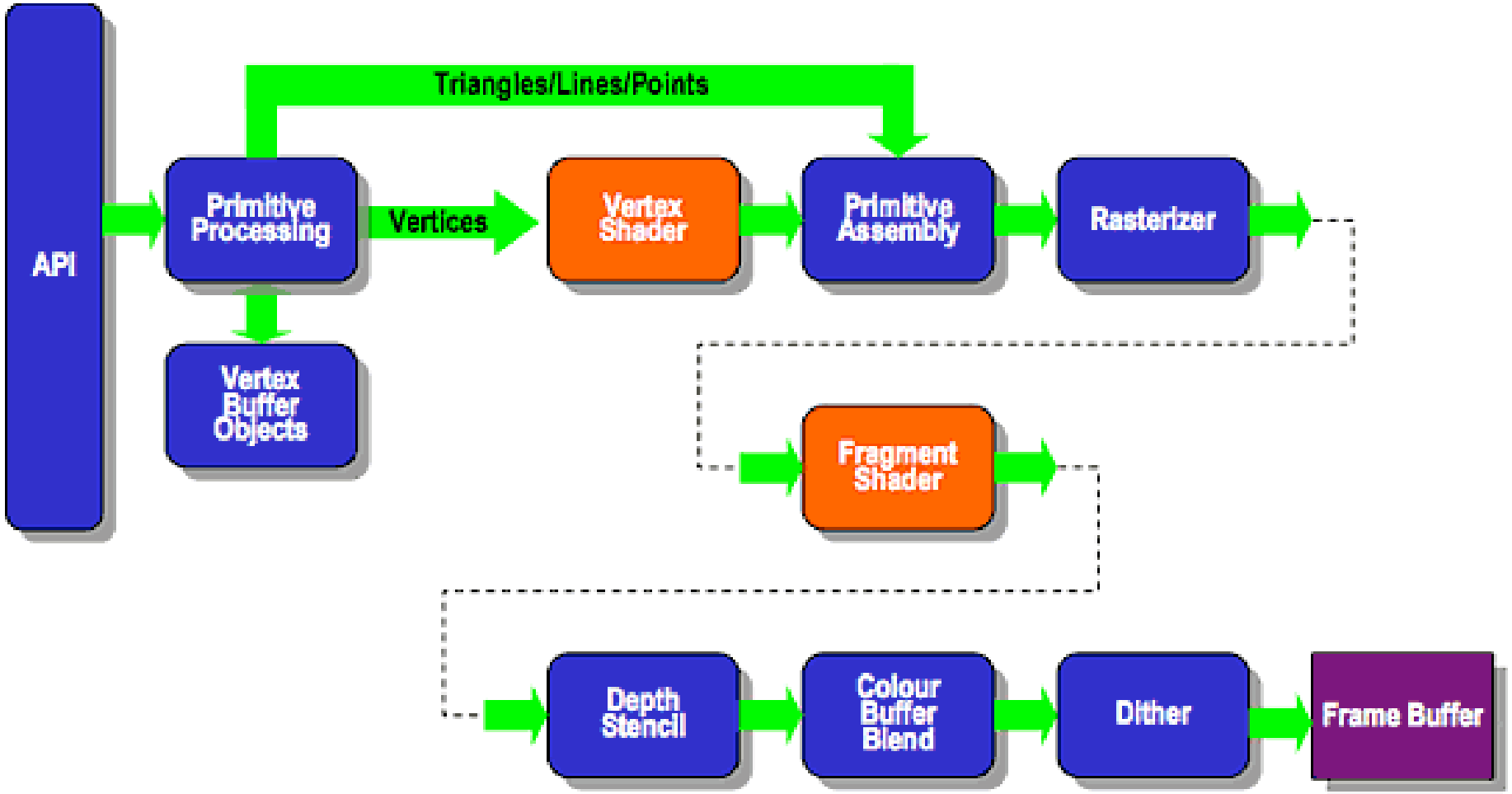
OpenGL Pipeline (Fixed-Function)

KB-VIS
We Work In-Depth



Programmable Pipeline

KB-VIS
We Work In-Depth



Fixed vs. Programmable

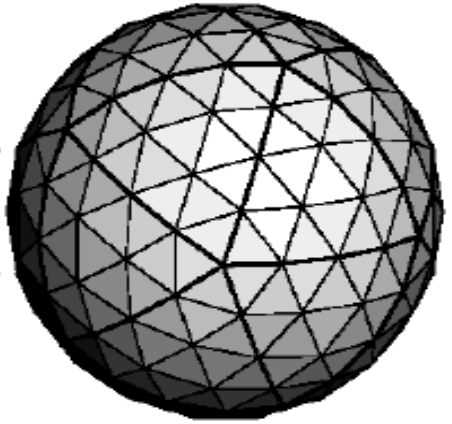
- Modern GPU architecture has rendered Fixed-Function pipeline obsolete
- Fixed-Function
  - separate hardware units for T&L, Texture etc.
  - dedicated API calls for each function
  - less scalable, inflexible - functionality enhanced only by adding new API calls, parameters, or GL states
- Modern GPUS
  - many general compute cores, massively parallel
  - graphics operations programmable using shaders
  - some functionality needs to be implemented by the programmer now (e.g. matrix transform, lighting & shading computations)

OpenGL State

- OpenGL is a State Machine
- 2D/3D Input Data

  (Vertex, Normal, Texture data)
- Configure state to modify functionality
  - Primitive Attributes (Colour, Position, Normals)
  - Light Settings
  - Texture Data and Environment
  - Transform State (Rotation, Translation, Scaling)
- Output to Framebuffer

  (Image on/off screen)



OpenGL Context

- Stores everything associated with the current OpenGL instance (state, framebuffer etc.)
- Multiple contexts possible
- Only one context *current* at a time in a thread
- OpenGL commands affect the state of whichever context is current
- A single context cannot be current in multiple threads at the same time

Creating an OpenGL Context

GLUT:

```
glutInit();
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize (250, 250);
glutInitWindowPosition (100, 100);
glutCreateWindow ("Hello World");
```

WGL:

```
PIXELFORMATDESCRIPTOR oPixelFormatDescriptor = {
        sizeof(PIXELFORMATDESCRIPTOR), 1,
        PFD_DRAW_TO_WINDOW | PFD_DOUBLEBUFFER | PFD_SUPPORT_OPENGL,
        PFD_TYPE_RGBA, 32, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 24, 0, 0,
        PFD_MAIN_PLANE, 0,0, 0, 0
        };

        int nPixelFormat = ChoosePixelFormat(hDC, &oPixelFormatDescriptor);
        DescribePixelFormat(hDC, nPixelFormat, sizeof(PIXELFORMATDESCRIPTOR),
                                                        &oPixelFormatDescriptor);
        SetPixelFormat(hDC, nPixelFormat, &oPixelFormatDescriptor);

        HGLRC hContext = wglCreateContext(hDC);
        wglMakeCurrent(hDC, hContext);
```

Getting Started – The Old Way


```
#include <whateverYouNeed.h>

main()
{
     InitializeOpenGLWindow();
     glClearColor (0.0, 0.0, 0.0, 0.0);
     glClear (GL_COLOR_BUFFER_BIT);
     glColor3f (1.0, 1.0, 1.0);

     glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);

     glBegin(GL_POLYGON);
       glVertex3f (0.25, 0.25, 0.0);
       glVertex3f (0.75, 0.25, 0.0);
       glVertex3f (0.75, 0.75, 0.0);
       glVertex3f (0.25, 0.75, 0.0);
     glEnd();
}
```

Example 1.1 (OpenGL Red Book)

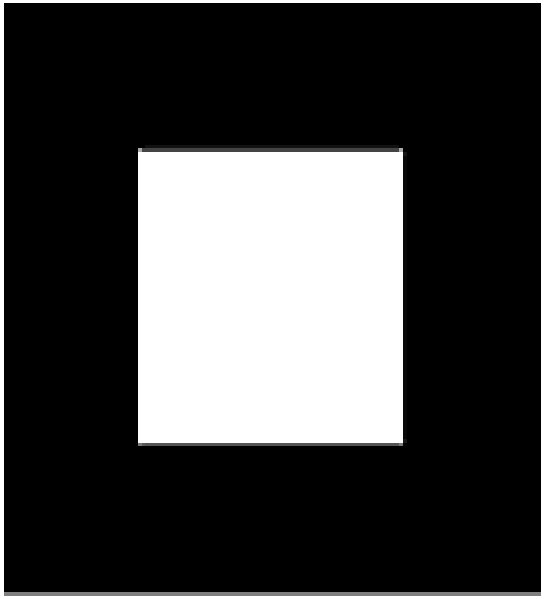


Figure 1-1 : White Rectangle on a Black Background

WebGL Context


```
rapiscanViewer.prototype.initialize = function () {

    this.canvas = document.createElement("canvas");
    this.canvas.width = this.screenDim;
    this.canvas.height = this.screenDim;
    this.context = this.canvas.getContext('webgl2',
                                { antialias: true });
    this.context.viewportWidth = this.canvas.width;
    this.context.viewportHeight = this.canvas.height;
    this.initGraphics(this.context);
};
```

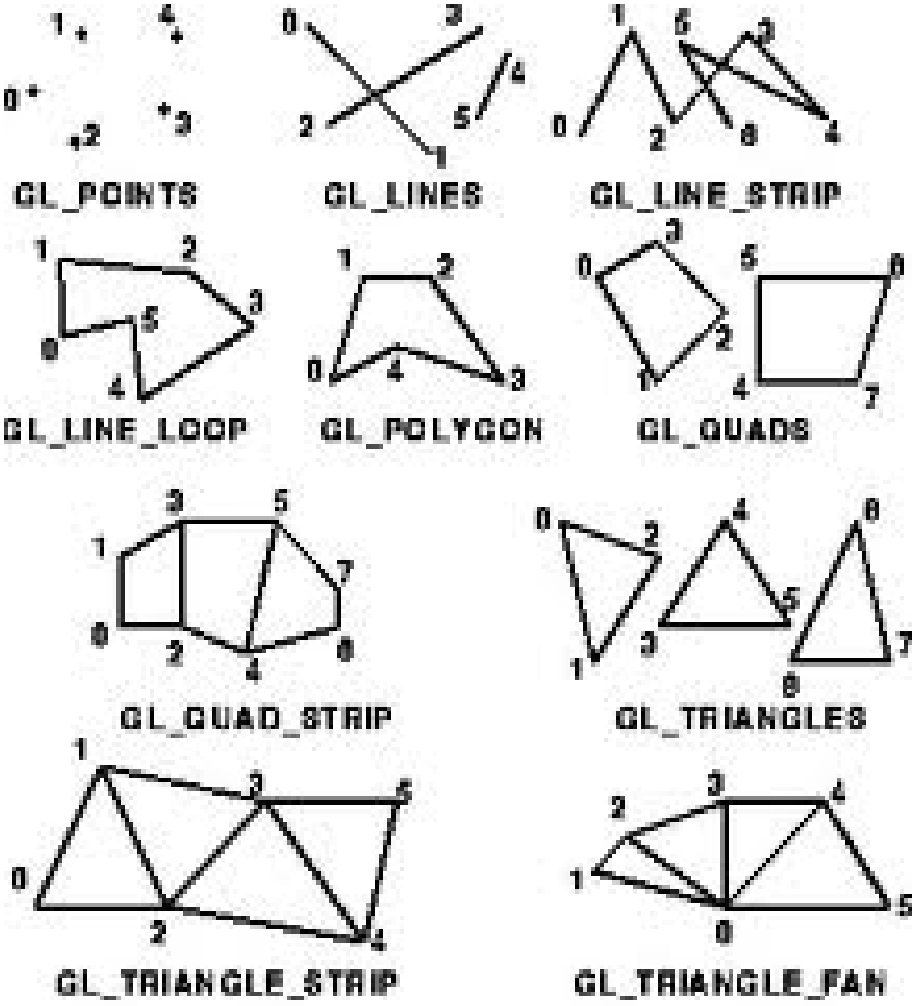
OpenGL Types

Type	Description
GLenum	Indicates that one of OpenGL's preprocessor definitions is expected.
GLboolean	Used for boolean values.
GLbitfield	Used for bitfields.
GLvoid	sed to pass pointers.
GLbyte	1-byte signed integer.
GLshort	GLshort 2-byte signed integer.
GLint	4-byte signed integer.
GLubyte	1-byte unsigned integer.
GLushort	2-byte unsigned integer.
GLuint	4-byte unsigned integer.
GLsizei	Used for sizes.
GLfloat	Single precision floating point number.
GLclampf	Single precision floating point number ranging from 0 to 1.
GLdouble	Double precision floating point number.
GLclampd	Double precision floating point number ranging from 0 to 1.

Drawing Primitives

- **Old Fixed-Function API:**

```
glBegin(GL_PRIMITIVE);
glVertex(x0,y0,z0);
glVertex(x1,y1,z1);
...
glEnd();
```

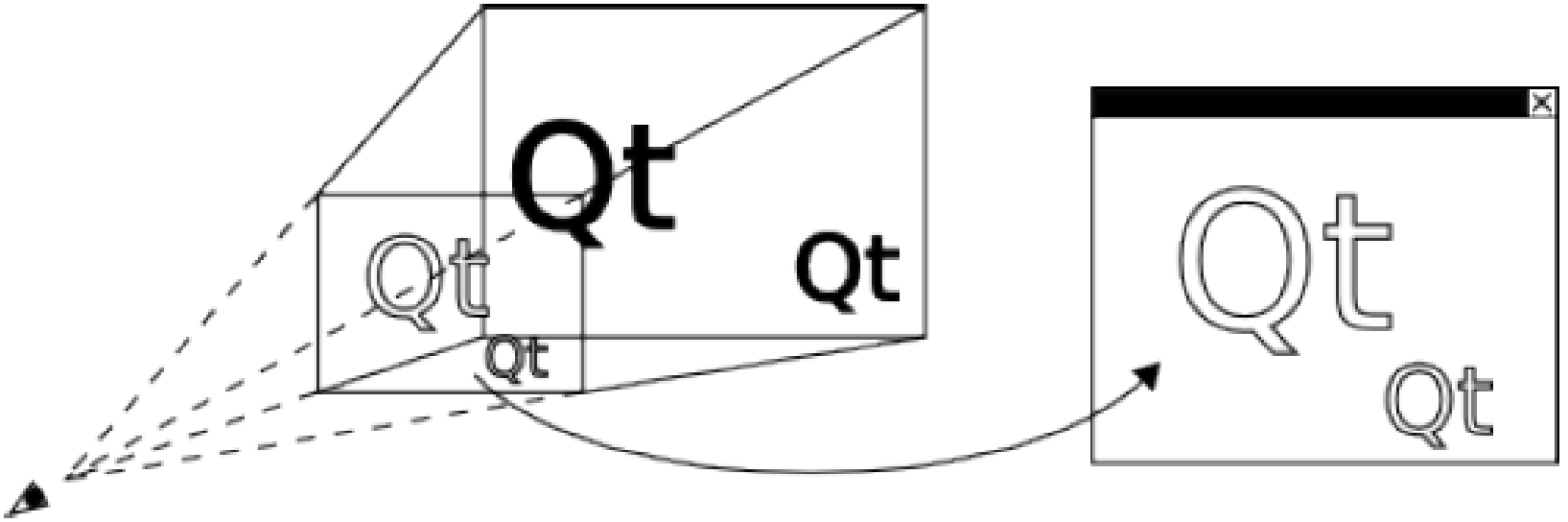


- **Current API:**

```
glDrawArrays(GLenum mode, GLint
   first, GLsizei count);
glDrawElements(GLenum
   mode,GLsizei count,GLenum
   type,const GLvoid *indices);
```

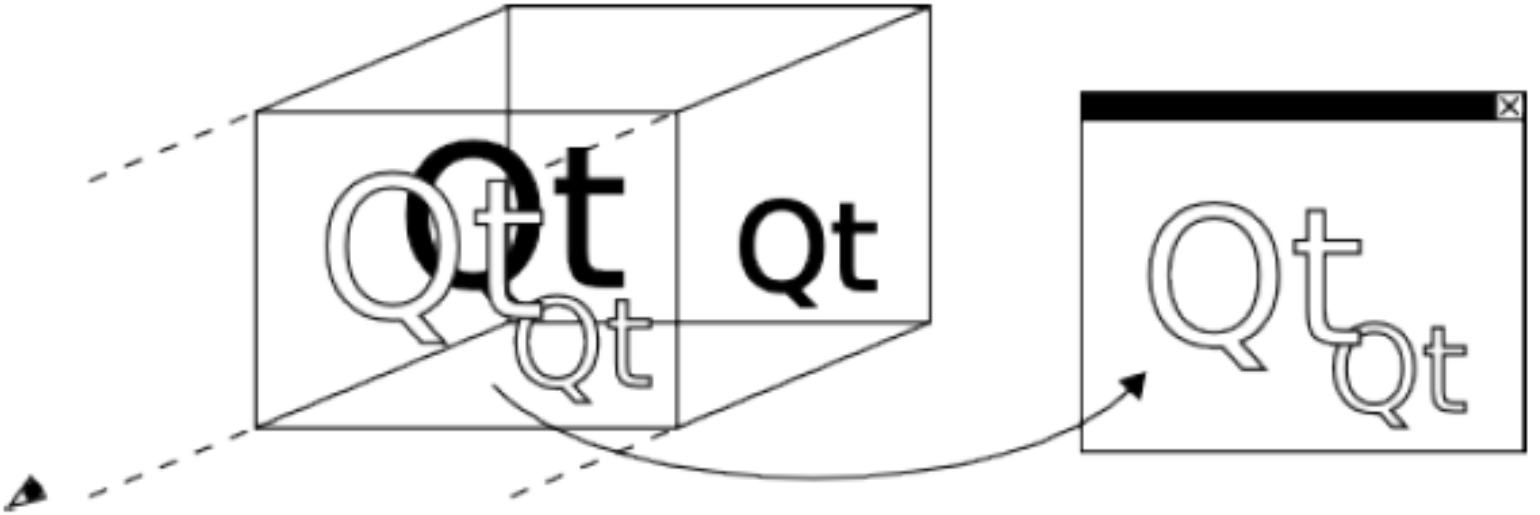
QUADS, POLYGONS not supported – use TRIANGLES instead

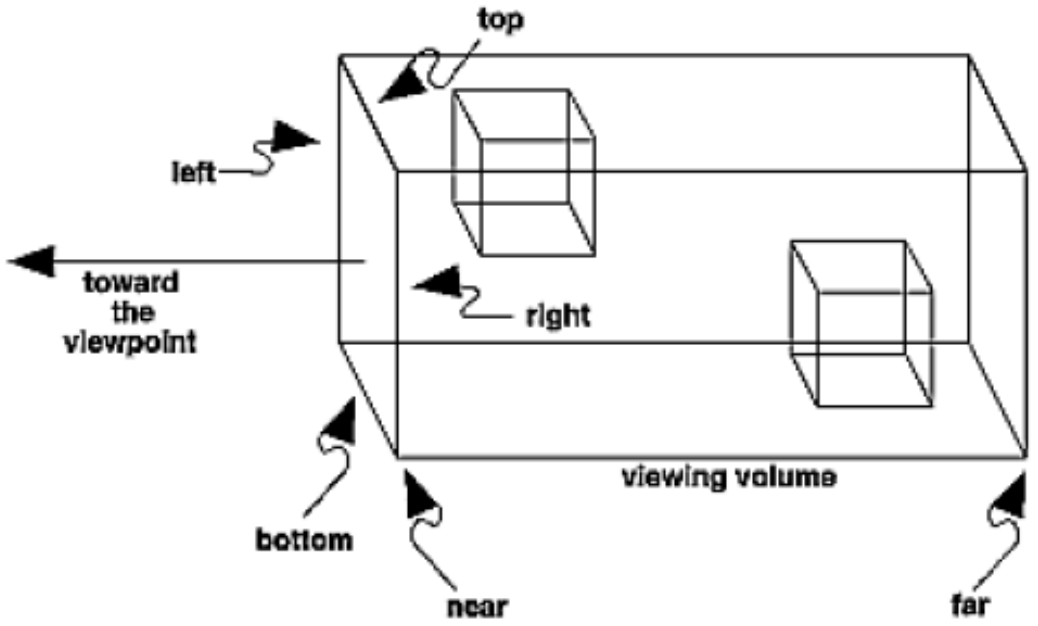
Perspective Projection



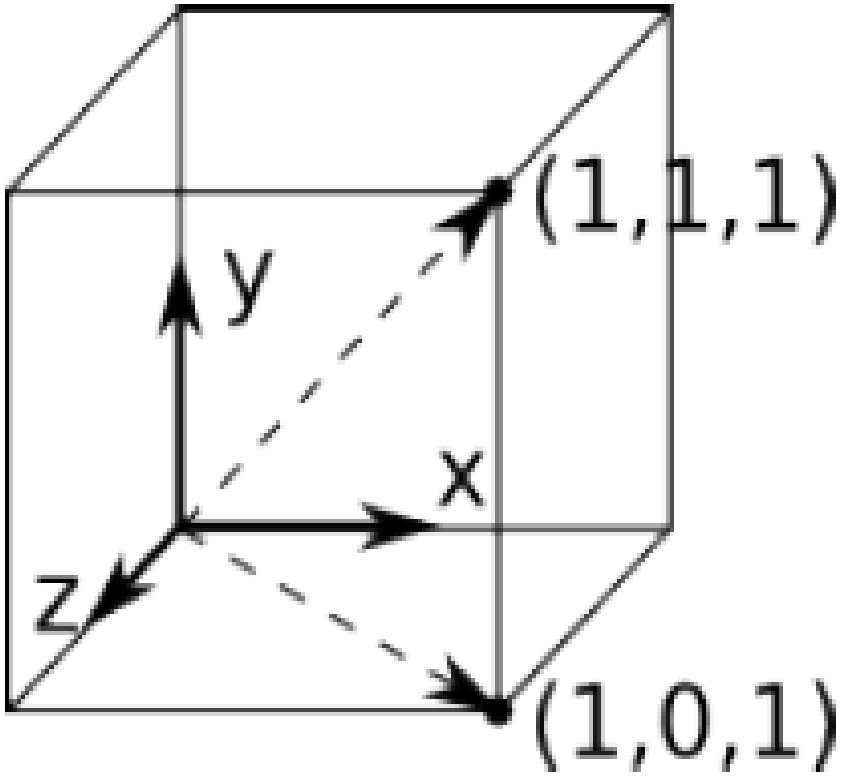
Orthographic Projection

KB-VIS
We Work In-Depth



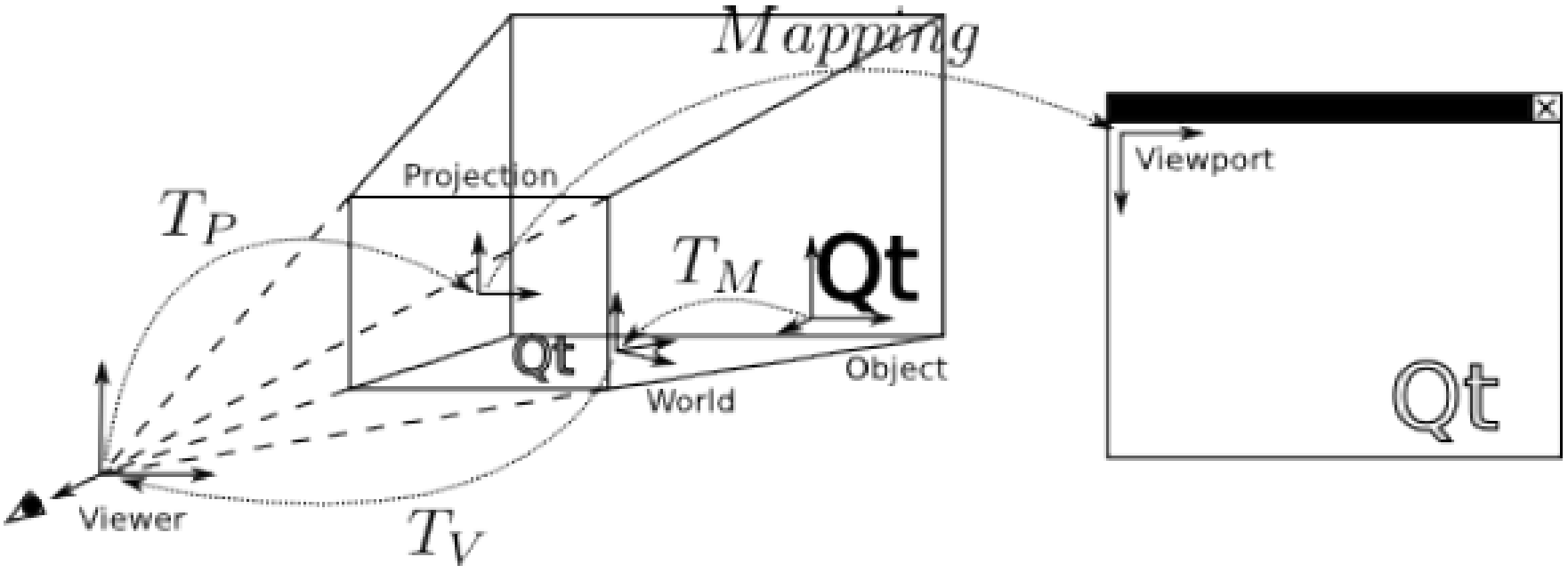


Coordinate System (Right-Handed)



v = (x,y,z)

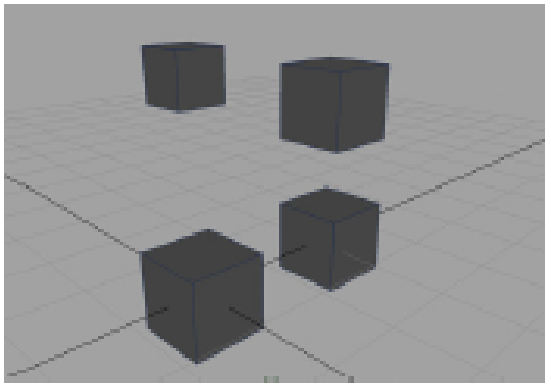
Object(3D) →View(2D) Mapping



Transformations

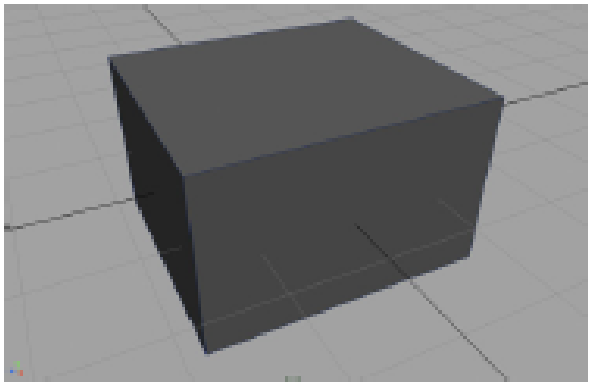
Translation

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



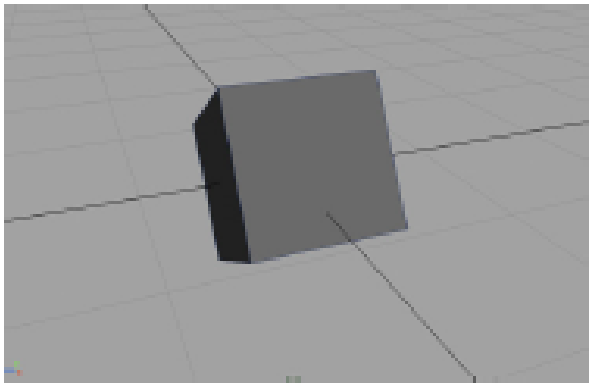
Scaling

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



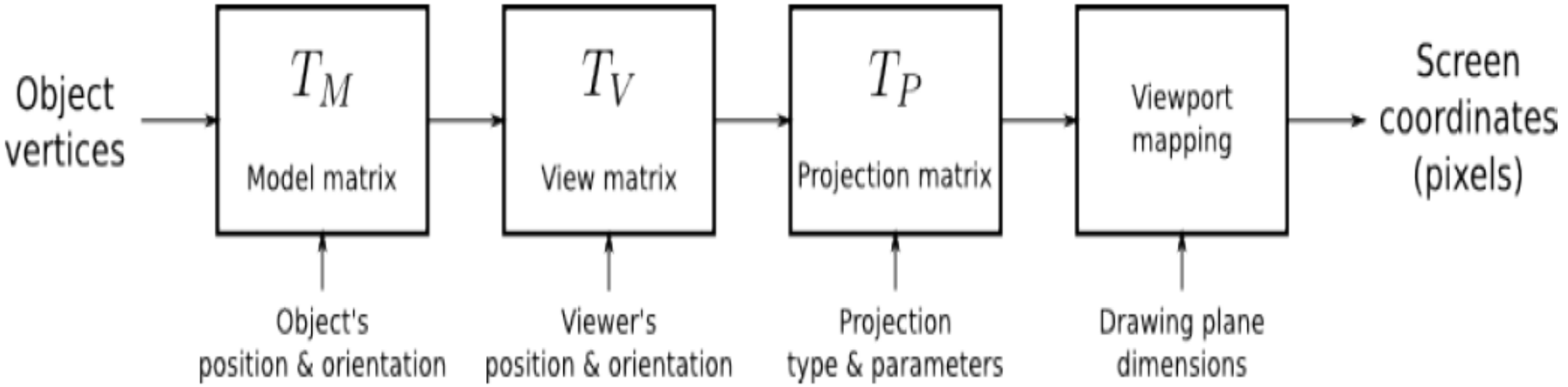
Rotation about z axis

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



$$x' = T \cdot x$$

Vertex ⟶ Pixel



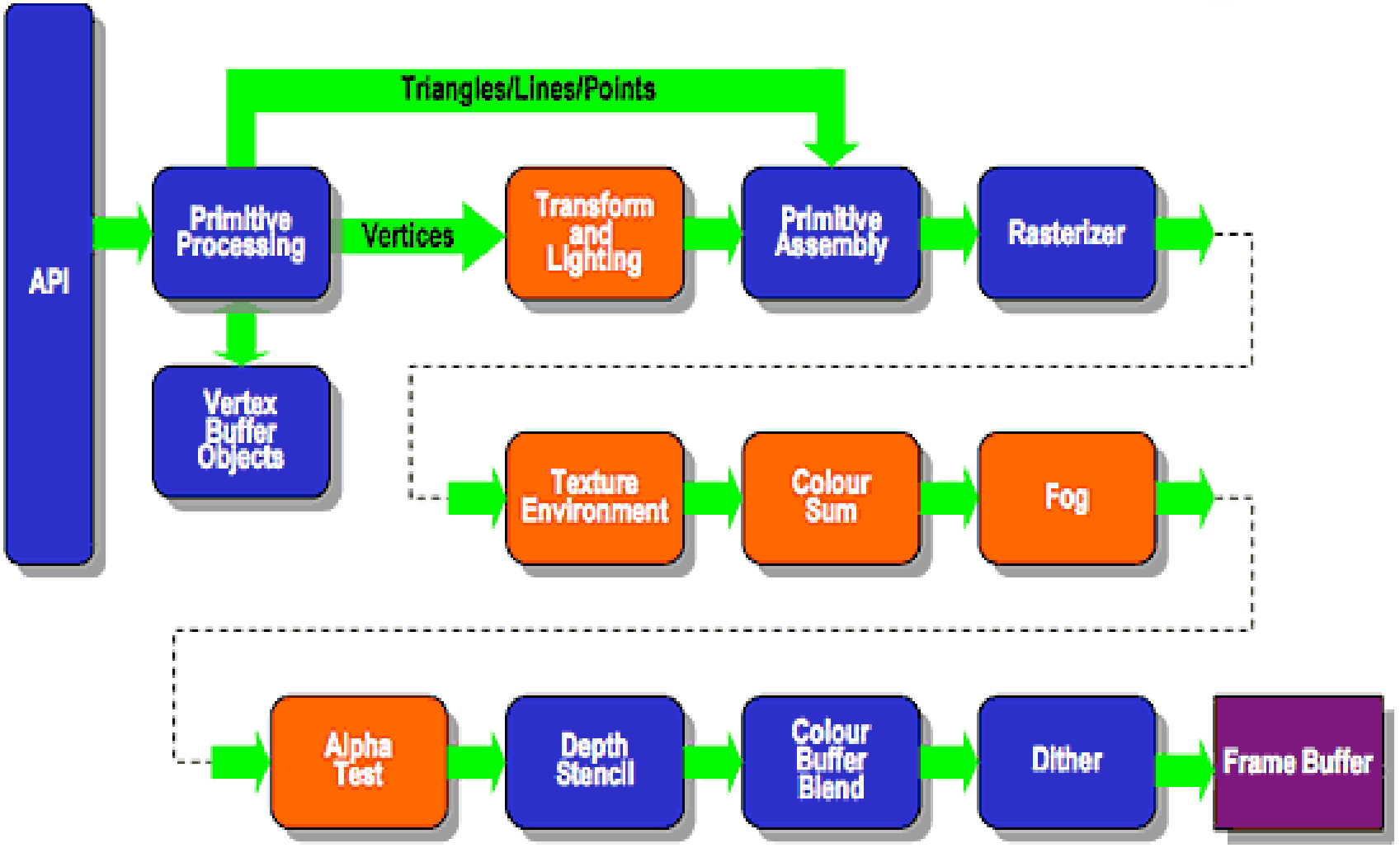
$$T = T_P * T_V * T_M$$

Note: multiply from the left

matrix multiplication is not commutative!

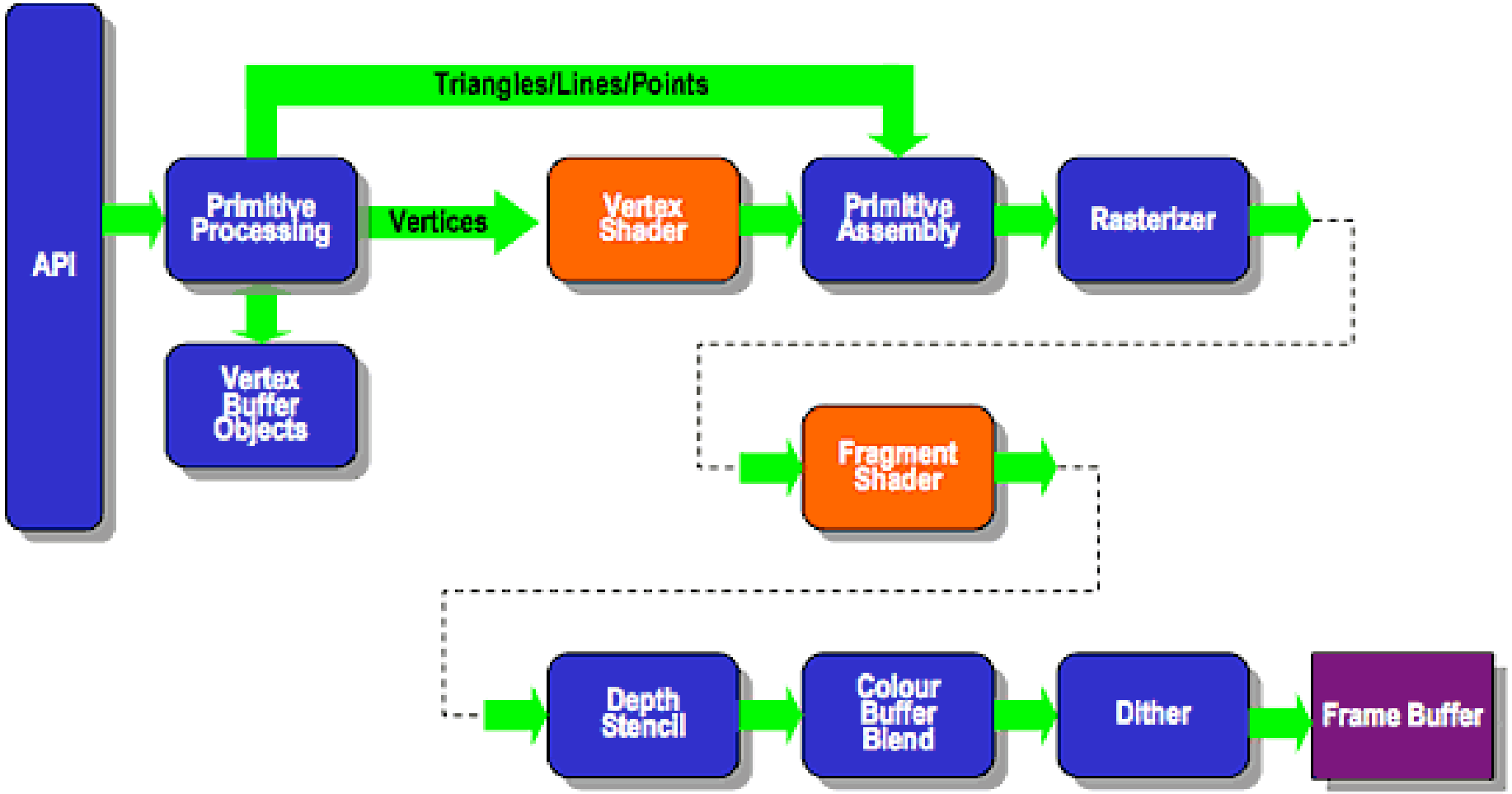
OpenGL Pipeline (Fixed-Function)

KB-VIS
We Work In-Depth



Programmable Pipeline

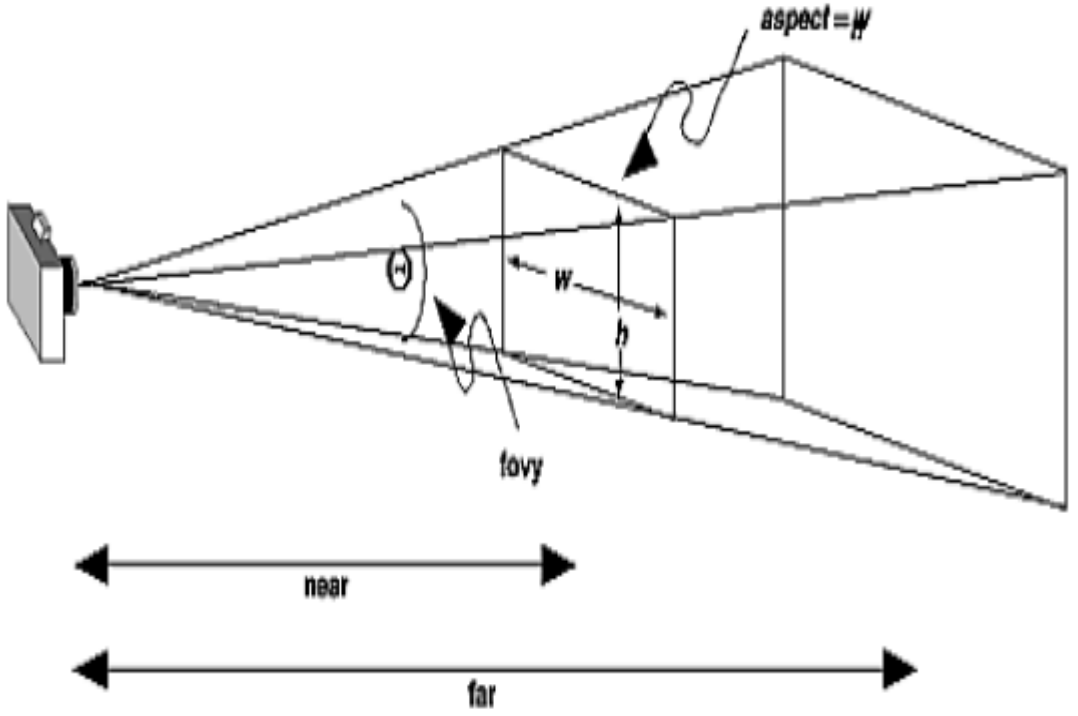
KB-VIS
We Work In-Depth



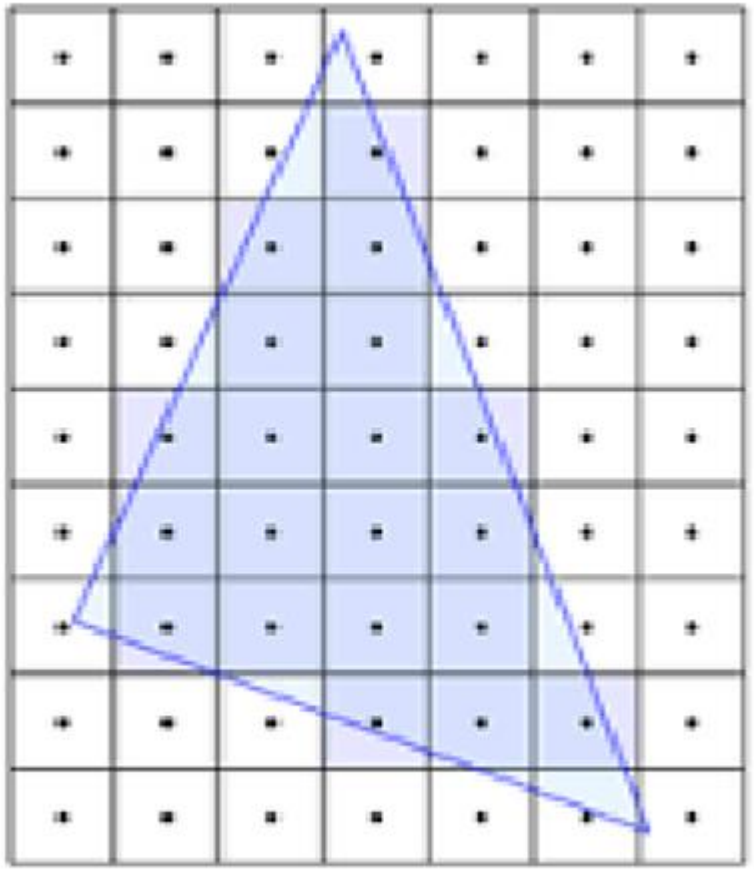
Viewport setup

```
function initView() {
        gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
        mat4.perspective(45, aspectRatio, 0.1, 100.0, pMatrix);
        mat4.identity(mvMatrix);
        setMatrixUniforms();
}
```

Viewport Mapping
transforms Camera
Coordinates (normalized)
to
Window Coordinates
(pixel units)



Rasterization



- Frame buffer is subdivided into a grid of pixels
- Determine which pixels are covered by the primitive
- Fragment – data relevant to a pixel, finally resolves to a colour/output value (RGB, luminance etc.)

Shaders

Vertex Shader:

```
uniform mat4 mvpMatrix;
in vec4 vertex;
in vec4 color;
out vec4 varyingColor;

void main(void)
{
	varyingColor = color;
	gl_Position = mvpMatrix * vertex;
}
```

Fragment Shader:

```
in vec4 varyingColor;
out vec4 fragColor;

void main(void)
{
	fragColor = varyingColor;
}
```

WebGL Setup

- WebGL Examples:

http://kbvis.com/downloads/kbvis-examples-webgl.zip

- WebGL Inspector:

http://benvanik.github.io/WebGL-Inspector/

- Latest version of Google Chrome:

https://www.google.com/intl/en/chrome/browser/

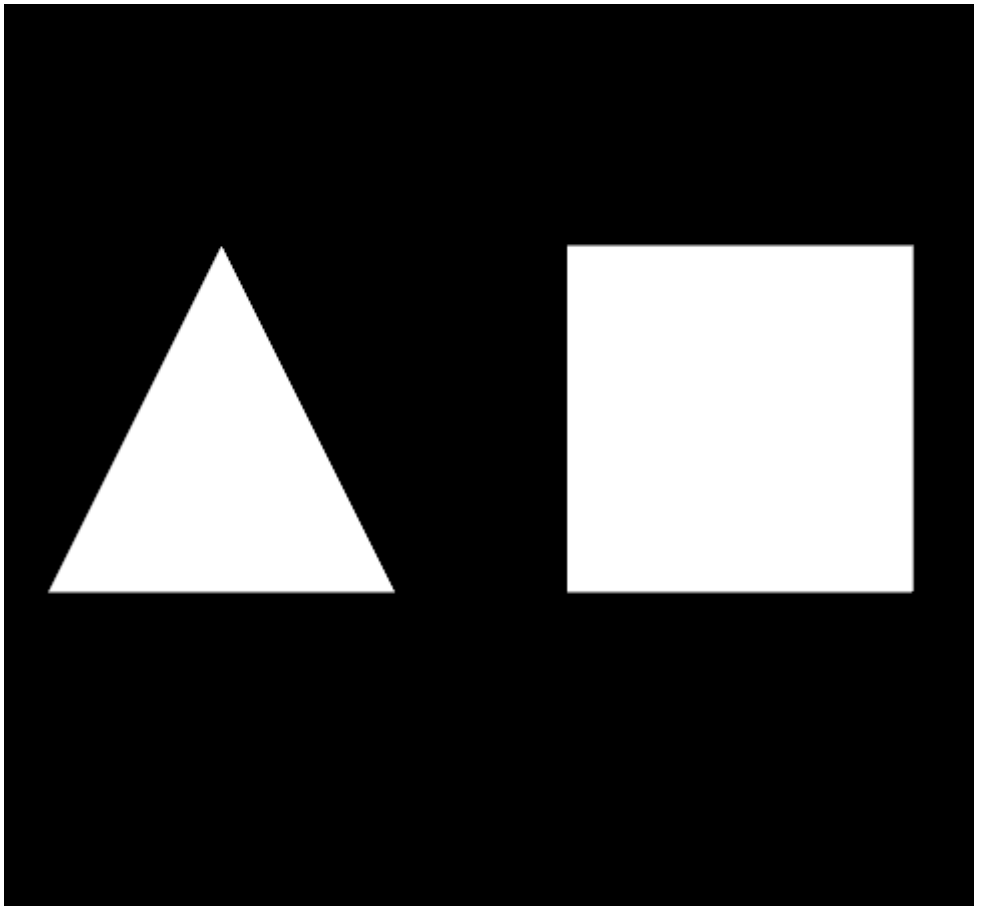
- Launch chrome.exe with
- "*--allow-file-access-from-files*" option to load textures and models locally. (As this undermines browser security, do NOT use this mode for browsing or deployment)
- "*--enable-unsafe-es3-apis*" option to enable WebGL 2.0

DAY 2

Getting Started

- Create a WebGL context
- Implement vertex and fragment shaders
- Load your shader source code through the WebGL API
- Compile and link your shaders
- Load your vertex data into the WebGL buffers
- Use the buffers to draw your scene
- Debug and troubleshoot your application

Hello WebGL!



See examples/hello-opengl

Hello WebGL - Source


```
<html>
<head>

<script id="shader-fs" type="x-shader/x-fragment">
    ...
</script>

<script id="shader-vs" type="x-shader/x-vertex">
    ...
</script>

<script type="text/javascript">

    var gl;
    function initGL(canvas) {
        ...
    }

    function getShader(gl, id) {
        ...
    }

    var shaderProgram;

    function initShaders() {
        ...
    }

    function initBuffers() {
        ...
    }

    function drawScene() {
        ...
    }

    function webGLStart() {
        var canvas = document.getElementById("myCanvas");
        ...
    }
</script>

</head>

<body onload="webGLStart();">
    <canvas id="myCanvas" style="border: none;" width="500" height="500"></canvas>
</body>

</html>
```

Creating the WebGL Context

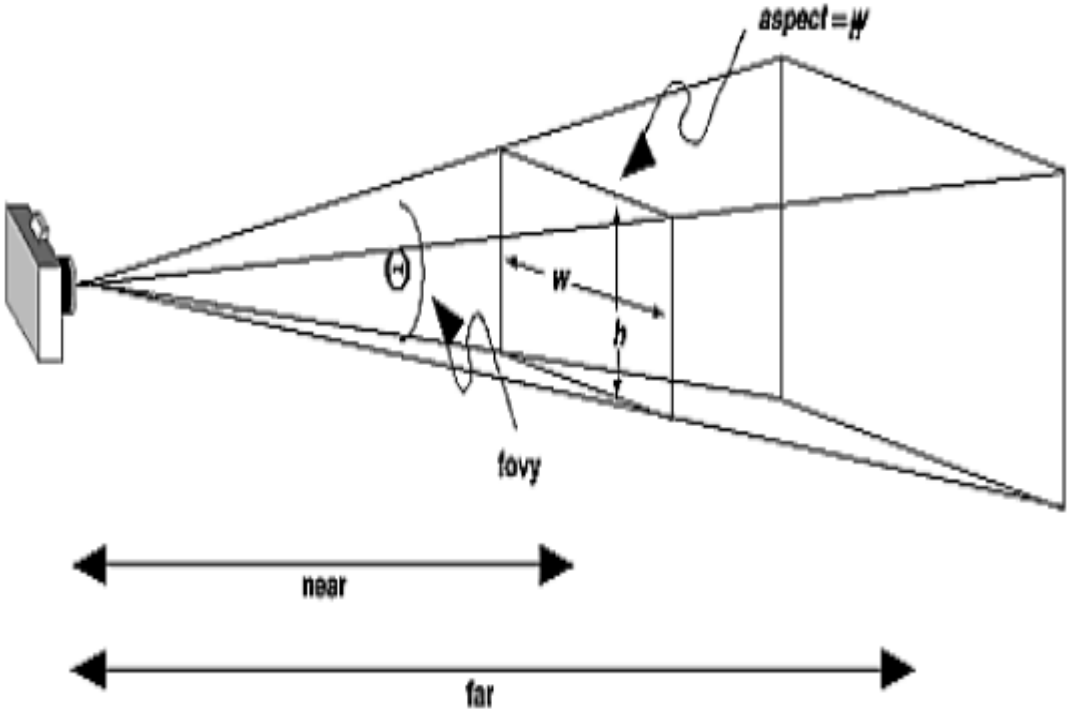

```
<body onload="webGLStart();">
    <canvas id="myCanvas" style="border: none;"
                                    width="500" height="500"></canvas>
</body>

function webGLStart() {
        var canvas = document.getElementById("myCanvas");
        initGL(canvas);
        initShaders();
        initBuffers();

        gl.clearColor(0.0, 0.0, 0.0, 1.0);
        gl.enable(gl.DEPTH_TEST);

        drawScene();
}
```

- Include the <canvas> tag in the HTML code
- Create a reference to your canvas so you can create a `WebGLRenderingContext`
- The `onload` event is triggered when the webpage is loaded
- `webGLStart()` is the entry point into your WebGL application

Viewport Setup


```
gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0, pMatrix);
```

Viewport Mapping transforms
Camera Coordinates
(normalized)
to
Window Coordinates
(pixel units)



glViewport

Name

glViewport — set the viewport

C Specification

```
void glViewport( GLint x,
                 GLint y,
                 GLsizei width,
                 GLsizei height);
```

Parameters

x, *y*

Specify the lower left corner of the viewport rectangle, in pixels. The initial value is (0,0).

width, *height*

Specify the width and height of the viewport. When a GL context is first attached to a window, *width* and *height* are set to the dimensions of that window.

www.khronos.org/registry/OpenGL-Refpages/

Creating Geometry

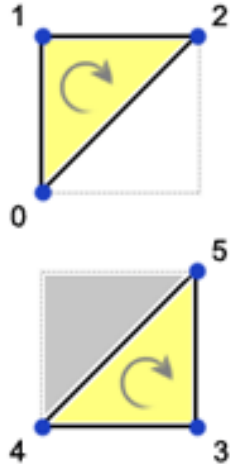

```
function initBuffers() {
        triangleVertexPositionBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
        var vertices = [
             0.0,  1.0,  0.0,
            -1.0, -1.0,  0.0,
             1.0, -1.0,  0.0
        ];
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
        triangleVertexPositionBuffer.itemSize = 3;
        triangleVertexPositionBuffer.numItems = 3;

        squareVertexPositionBuffer = gl.createBuffer();
        gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
        vertices = [
             1.0,  1.0,  0.0,
            -1.0,  1.0,  0.0,
             1.0, -1.0,  0.0,
            -1.0, -1.0,  0.0
        ];
        gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
        squareVertexPositionBuffer.itemSize = 3;
        squareVertexPositionBuffer.numItems = 4;
    }
```

- Create the Vertex Buffer using `createBuffer()`
- Bind the Vertex Buffer using `bindBuffer()` . Subsequent buffer API calls apply to this buffer
- Allocate and populate the buffer using `bufferData()`

Vertex and Index Buffers

Vertex Buffer

VB Index :	
0	(-1, -1)
1	(-1, 1)
2	(1, 1)
3	(1, -1)
4	(-1, -1)
5	(1, 1)

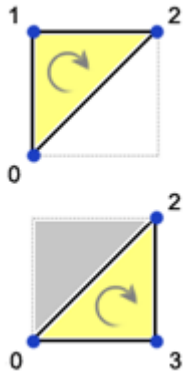


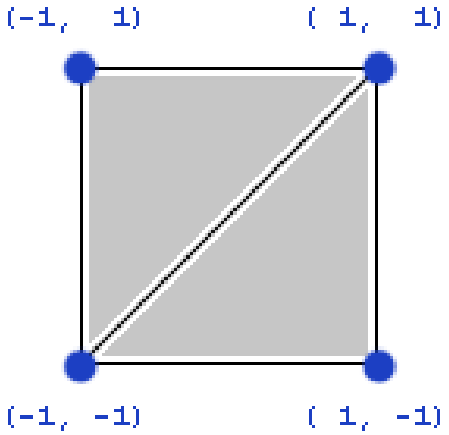
Index Buffer

IB Index :	
0	0
1	1
2	2
3	3
4	0
5	2

Vertex Buffer

VB Index :	
0	(-1, -1)
1	(-1, 1)
2	(1, 1)
3	(1, -1)





with Indexing

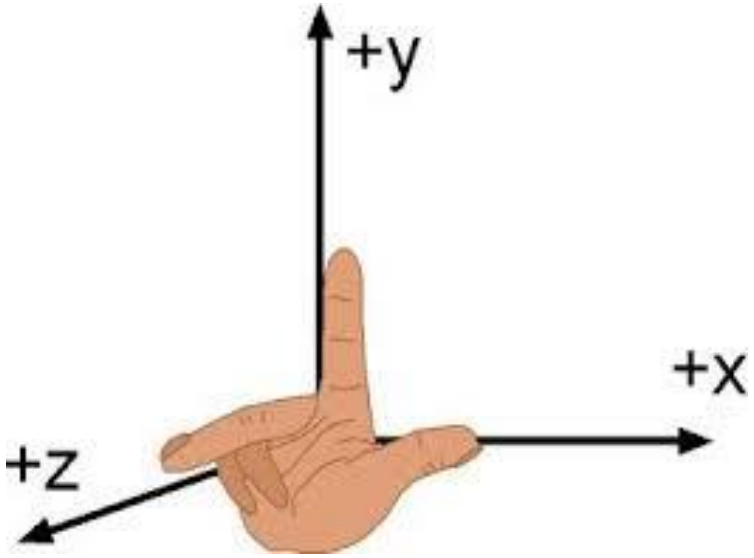
Drawing the Scene

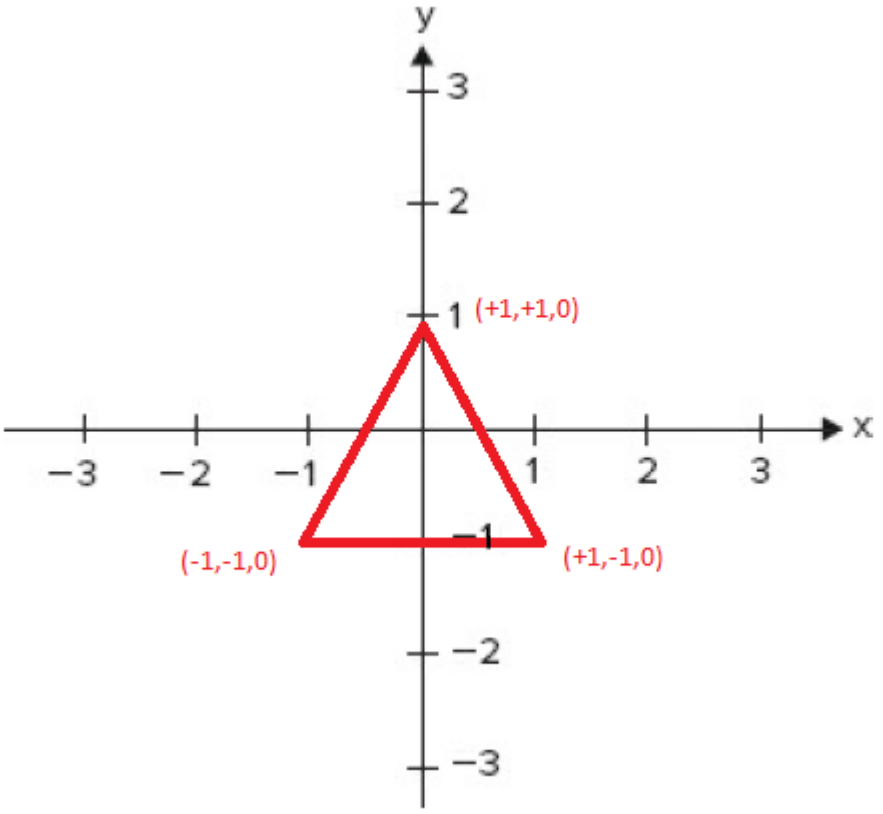

```
    function drawScene() {
        gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
        gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

        mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0, pMatrix);

        mat4.identity(mvMatrix);

        mat4.translate(mvMatrix, [-1.5, 0.0, -7.0]);
        gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
        gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
                               triangleVertexPositionBuffer.itemSize, gl.FLOAT,
false, 0, 0);
        setMatrixUniforms();
        gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);


        mat4.translate(mvMatrix, [3.0, 0.0, 0.0]);
        gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
        gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
                               squareVertexPositionBuffer.itemSize, gl.FLOAT,
false, 0, 0);
        setMatrixUniforms();
        gl.drawArrays(gl.TRIANGLE_STRIP, 0, squareVertexPositionBuffer.numItems);
    }
```

- Specify the viewport using `viewport()`
- Bind the vertex buffer, and link the shader vertex attribute to the buffer data using `vertexAttribPointer()`
- Draw the vertex arrays using `drawArrays()`

Positioning Objects


```
mat4.identity(mvMatrix);
mat4.translate(mvMatrix, [-1.5, 0.0, -7.0]);
...
setMatrixUniforms();
```





Right Handed Coordinate System

Drawing Vertices


```
gl.drawArrays(    mode, first, count );
```

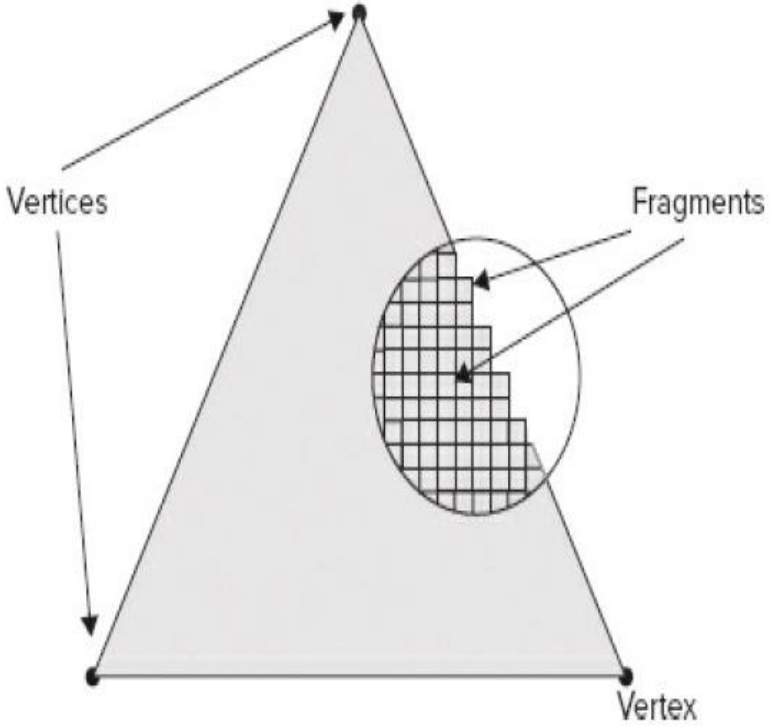
- render geometric primitives from bound and enabled vertex data
- *mode* [in] – type of primitives to render - gl.LINE_STRIP, gl.LINES, gl.POINTS, gl.TRIANGLE_STRIP, or gl.TRIANGLES
- *first* [in] - starting index in the enabled array
- *count* [in] - number of indices to be rendered

```
gl.drawElements( mode, count, type, offset );
```

- renders geometric primitives indexed by element array data
- *mode* [in] – type of primitives to render. gl.LINE_STRIP, gl.LINES, gl.POINTS, gl.TRIANGLE_STRIP, or gl.TRIANGLES
- *count* - number of indices to be rendered
- type [in] - type of elements. Must be gl.UNSIGNED_SHORT
- offset [in] - Offset into element array buffer

Vertex and Fragment Shaders


```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition
    }
</script>

<script id="shader-fs" type="x-shader/x-fragment">
    precision mediump float;

    void main(void) {
        gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
    }
</script>
```



- Vertex Shader – uses vertex attributes, transformation state, uniforms (global constant data) to compute vertex position and other values (normals, lighting values etc.)
- Fragment Shader – uses vertex data from the vertex shader (interpolated across fragments) and uniforms to compute the final colour of a fragment
- Computed values from the vertex shader are passed to the fragment shader via `varying` variables

Buffer Objects

- Transferring vertex data to GPU for every draw is inefficient
- VBOs are arrays of vertex data on the GPU
- PBOs are used to perform pixel transfers
- After allocation, buffers may be read and written to
- **gl.bufferData** - creates and initializes a buffer object's data store
- **gl.bindBuffer** - bind a buffer object for use

VBO

- **Vertex Buffer Object** - source for vertex array data
- **gl.vertexAttribPointer** - define an array of generic vertex attribute data, tell OpenGL programmable pipeline how to locate vertex position data
- **gl.drawArrays -** render vertices from array data
- **gl.drawElements -** specify vertices by using array of indices from an Index Buffer, avoid duplicating repeated vertices

glGenBuffers

glGenBuffers — generate buffer object names

C Specification

```
void glGenBuffers(GLsizei  n,
                  GLuint * buffers);
```

Parameters

n

Specifies the number of buffer object names to be generated.

buffers

Specifies an array in which the generated buffer object names are stored.

http://docs.gl/gl3/glGenBuffers

glBindBuffer

glBindBuffer — bind a named buffer object

C Specification

```
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, 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.

buffer

Specifies the name of a buffer object.

http://docs.gl/gl3/glBindBuffer

glBufferData

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

C Specification

```
void glBufferData(GLenum       target,
                  GLsizeiptr   size,
                  const GLvoid * data,
                  GLenum       usage);
```

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

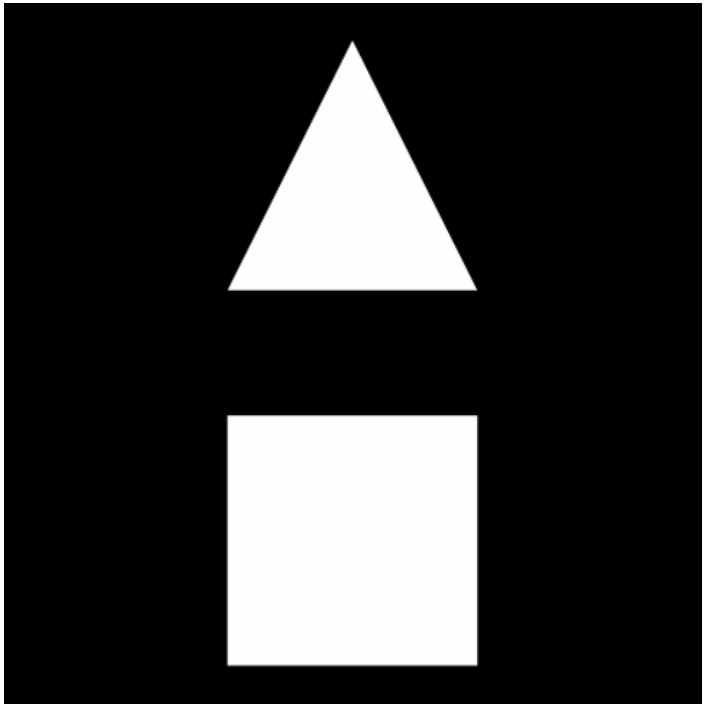
Specifies a pointer to data that will be copied into the data store for initialization, or NULL if no data is to be copied.

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.

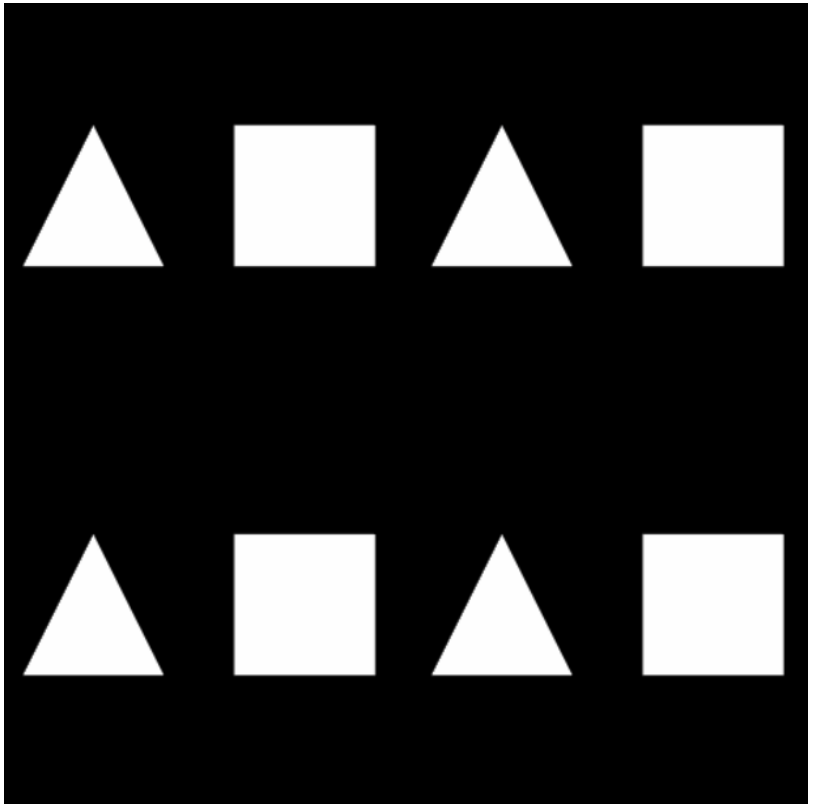
http://docs.gl/gl3/glBufferData

Exercise 1.1



- Centre the triangle above the square
- Send your edited index.html to:
- 3d@kbvis.com

Exercise 1.2



- Draw the objects four times,
in each of the four quadrants of the canvas
- Send your edited index.html to:
- 3d@kbvis.com

DAY 3

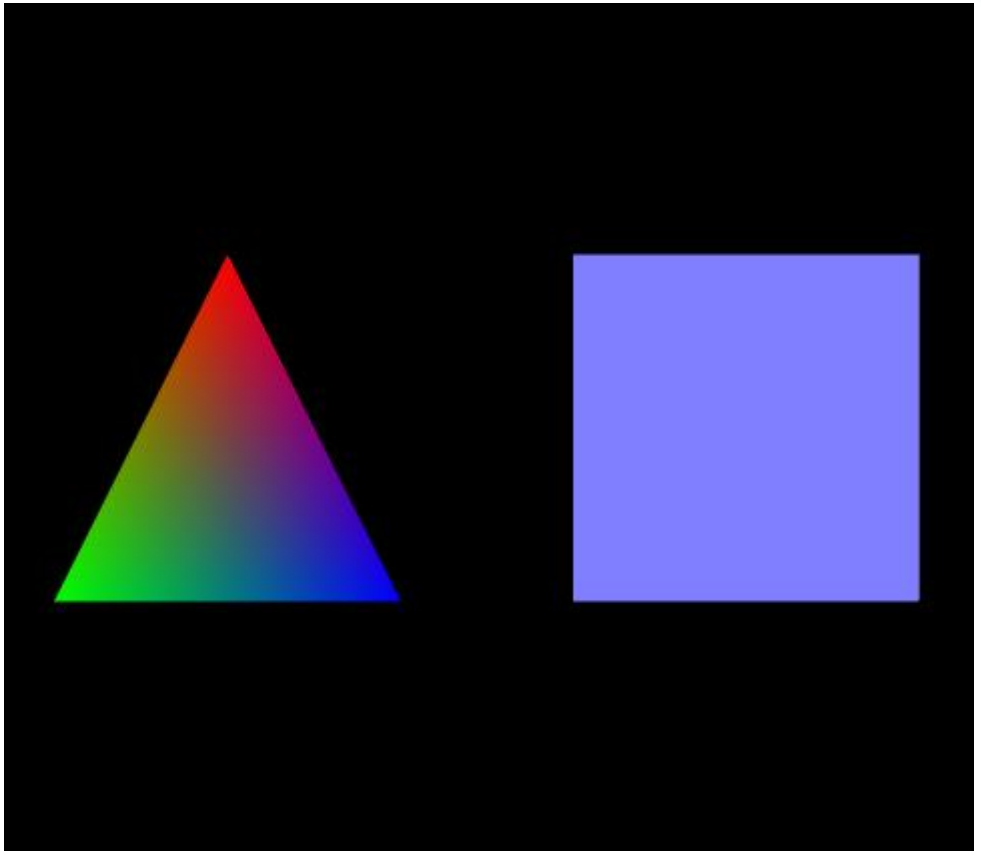
Exercise 1.1


```
function drawScene() {
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0, pMatrix);

    mat4.identity(mvMatrix);

    mat4.translate(mvMatrix, [0, 1.5, -7]);
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);

    mat4.translate(mvMatrix, [0.0, -3, 0.0]);
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, squareVertexPositionBuffer.numItems);
}
```

Exercise 1.2


```
function drawScene() {
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0, pMatrix);

    gl.viewport(0, 0, gl.viewportWidth/2, gl.viewportHeight/2);
    drawObjects();
    gl.viewport(gl.viewportWidth / 2, 0, gl.viewportWidth / 2, gl.viewportHeight / 2);
    drawObjects();
    gl.viewport(0, gl.viewportHeight / 2, gl.viewportWidth / 2, gl.viewportHeight / 2);
    drawObjects();
    gl.viewport(gl.viewportWidth / 2, gl.viewportHeight / 2, gl.viewportWidth / 2, gl.viewportHeight / 2);
    drawObjects();
}

function drawObjects(){
    mat4.identity(mvMatrix);
    mat4.translate(mvMatrix, [-1.5, 0.0, -7.0]);
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, triangleVertexPositionBuffer.itemSize, g
    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);

    mat4.translate(mvMatrix, [3.0, 0.0, 0.0]);
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, squareVertexPositionBuffer.itemSize, gl.
    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, squareVertexPositionBuffer.numItems);
}
```

Adding Colour



See example02-colour

Defining Colours


```
function initBuffers() {
        ...
    triangleVertexColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
    var colors = [
        1.0, 0.0, 0.0, 1.0,
        0.0, 1.0, 0.0, 1.0,
        0.0, 0.0, 1.0, 1.0,
    ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors), gl.STATIC_DRAW);
    triangleVertexColorBuffer.itemSize = 4;
    triangleVertexColorBuffer.numItems = 3;
        ...
    squareVertexColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
    colors = []
    for (var i=0; i < 4; i++) {
        colors = colors.concat([0.5, 0.5, 1.0, 1.0]);
    }
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors), gl.STATIC_DRAW);
    squareVertexColorBuffer.itemSize = 4;
    squareVertexColorBuffer.numItems = 4;
}
```

- Create the Colour attributes Buffer using `createBuffer()`
- Bind the Colour Buffer using `bindBuffer()`
- Allocate and populate the buffer using `bufferData()`

Setting up Vertex Attributes


```
function initShaders() {
    var fragmentShader = getShader(gl, "shader-fs");
    var vertexShader = getShader(gl, "shader-vs");

    shaderProgram = gl.createProgram();
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram(shaderProgram);

    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
        alert("Could not initialise shaders");
    }

    gl.useProgram(shaderProgram);

    shaderProgram.vertexPositionAttribute = gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.vertexColorAttribute = gl.getAttribLocation(shaderProgram, "aVertexColor");
    gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);

    shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram, "uMVMatrix");
}
```

gl.**getAttribLocation**() - returns the “address” of the specified shader variable
gl.**enableVertexAttribArray**() - the values in the vertex attribute array will be accessed and used for rendering when calls are made to glDrawArrays or glDrawElements
gl.**getUniformLocation**() - returns the “address” of the specified shader uniform, which can then be used to set the value.

Using Colour Attributes


```
function drawScene() {
        ...
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
            triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
     gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
            triangleVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);

        ...
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
            squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
     gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
            squareVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.drawArrays(gl.TRIANGLE_STRIP, 0, squareVertexPositionBuffer.numItems);
        ...
}
```

- Bind the colour buffer, and link the shader vertex attribute to the buffer data using `vertexAttribPointer()`
- Draw the vertex arrays using `drawArrays()`

glVertexAttribPointer

glVertexAttribPointer — define an array of generic vertex attribute data

C Specification

```
void glVertexAttribPointer(GLuint         index,
                           GLint          size,
                           GLenum         type,
                           GLboolean      normalized,
                           GLsizei        stride,
                           const GLvoid * pointer);
void glVertexAttribIPointer(GLuint         index,
                            GLint          size,
                            GLenum         type,
                            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, 4. Additionally, the symbolic constant GL_BGRA is accepted by `glVertexAttribPointer`. The initial value is 4.

type

Specifies the data type of each component in the array. The symbolic constants GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_INT, and GL_UNSIGNED_INT are accepted by both functions. Additionally GL_HALF_FLOAT, GL_FLOAT, GL_DOUBLE, GL_INT_2_10_10_10_REV, and GL_UNSIGNED_INT_2_10_10_10_REV are accepted by `glVertexAttribPointer`. The initial value is GL_FLOAT.

normalized

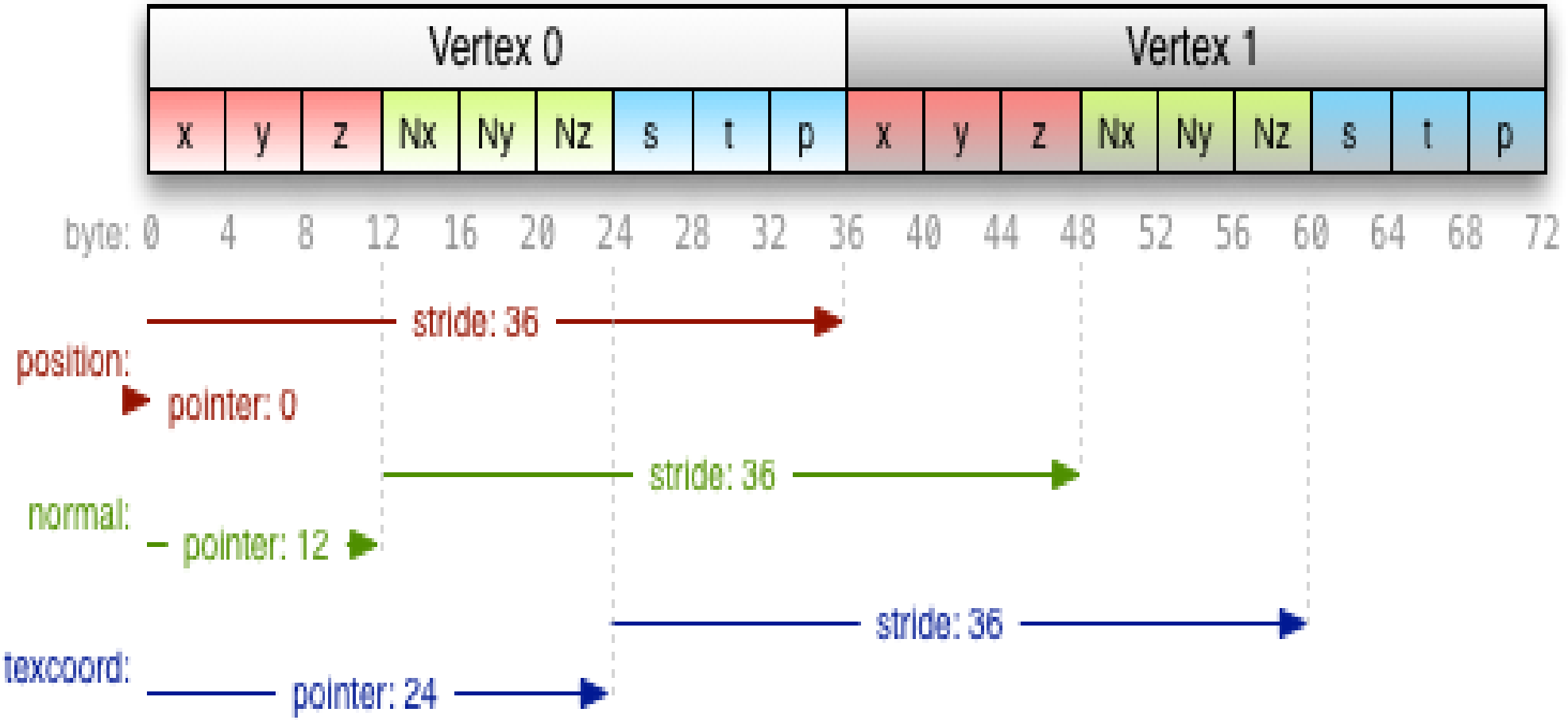
For `glVertexAttribPointer`, specifies whether fixed-point data values should be normalized (GL_TRUE) or converted directly as fixed-point values (GL_FALSE) when they are accessed.

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

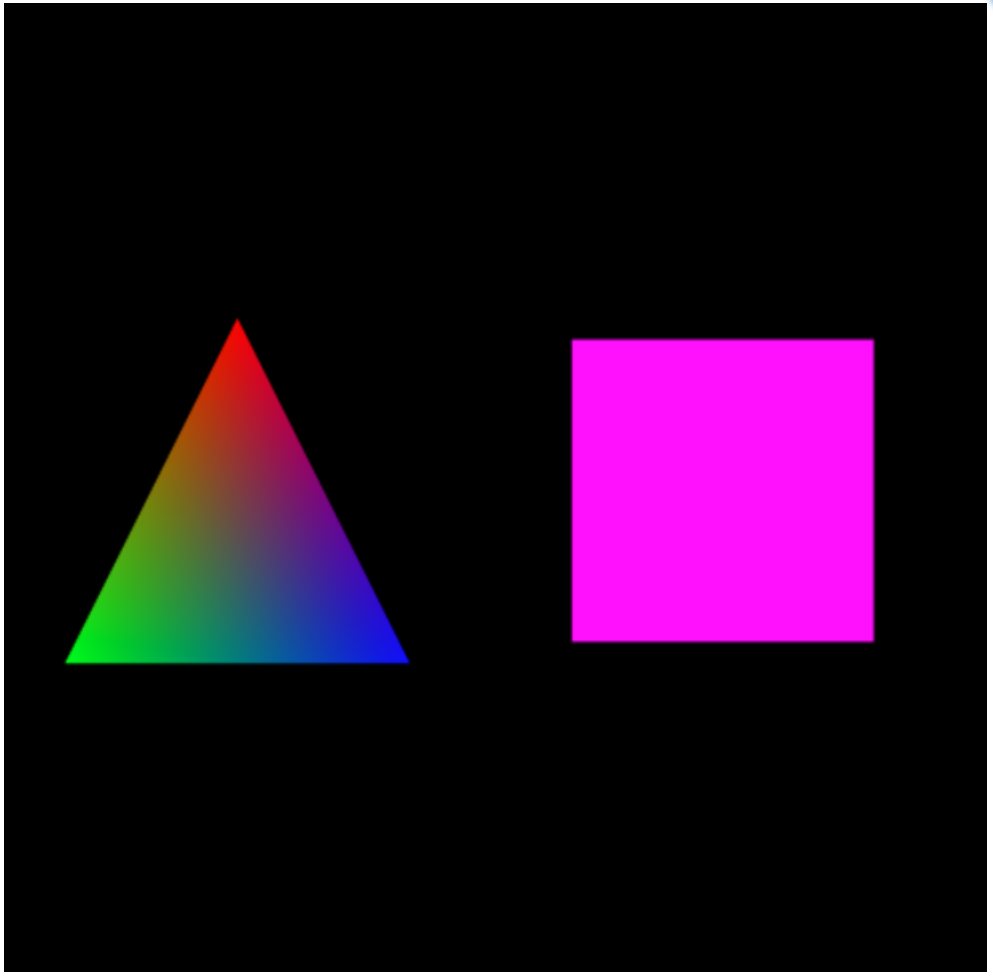
Specifies a offset of the first component of the first generic vertex attribute in the array in the data store of the buffer currently bound to the GL_ARRAY_BUFFER target. The initial value is 0.


Vertex 0
Vertex 1
x y z Nx Ny Nz s t p
x y z Nx Ny Nz s t p
byte: 0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
stride: 36
position:
pointer: 0
stride: 36
normal:
pointer: 12
stride: 36
texcoord:
pointer: 24

```
<script id="shader-vs" type="x-shader/x-vertex">#version 300 es
    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    in vec3 aVertexPosition;
    in vec4 aVertexColor;

    out vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = aVertexColor;
    }
</script>
<script id="shader-fs" type="x-shader/x-fragment">#version 300 es
    precision mediump float;

    in vec4 vColor;
    out vec4 fragColor;

    void main(void) {
        fragColor = vColor;
    }
</script>
```

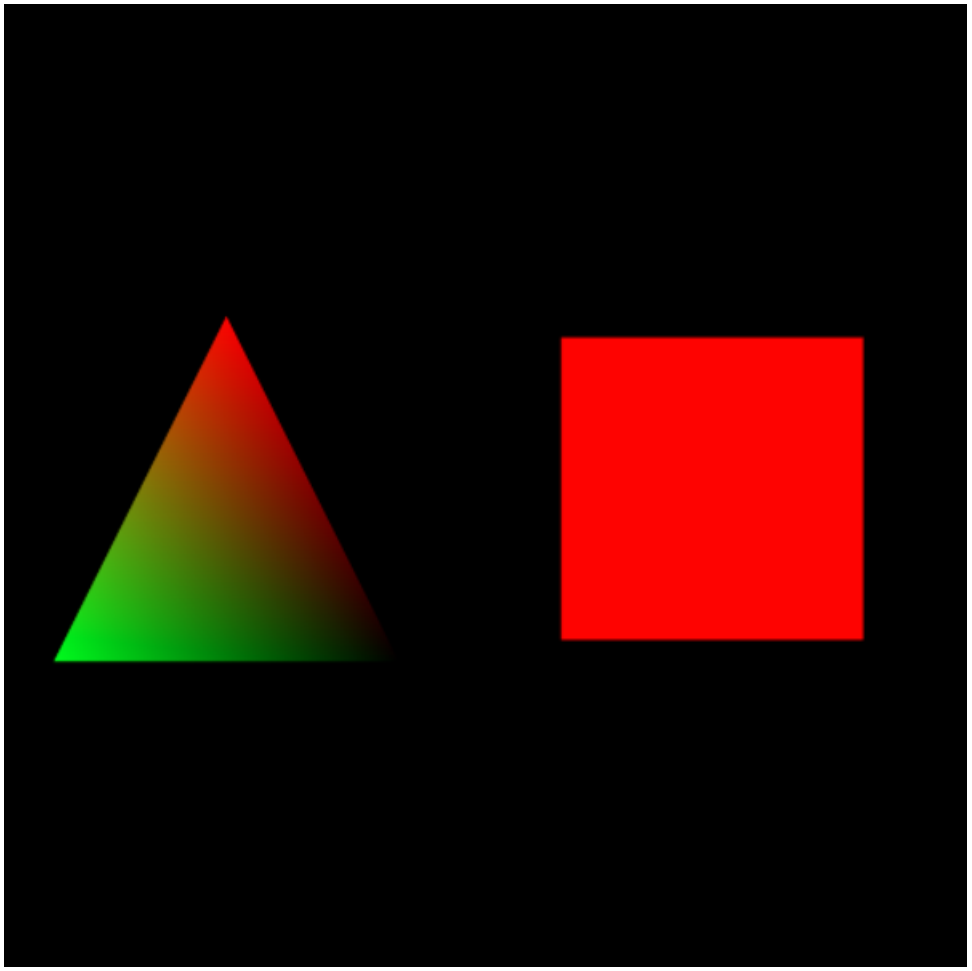
For more information on manipulating color vectors in shaders:
https://www.khronos.org/opengl/wiki/Data_Type_(GLSL)#Vectors

Exercise 2.1



- Change the colour of the square to Magenta (RED+BLUE)

Exercise 2.2

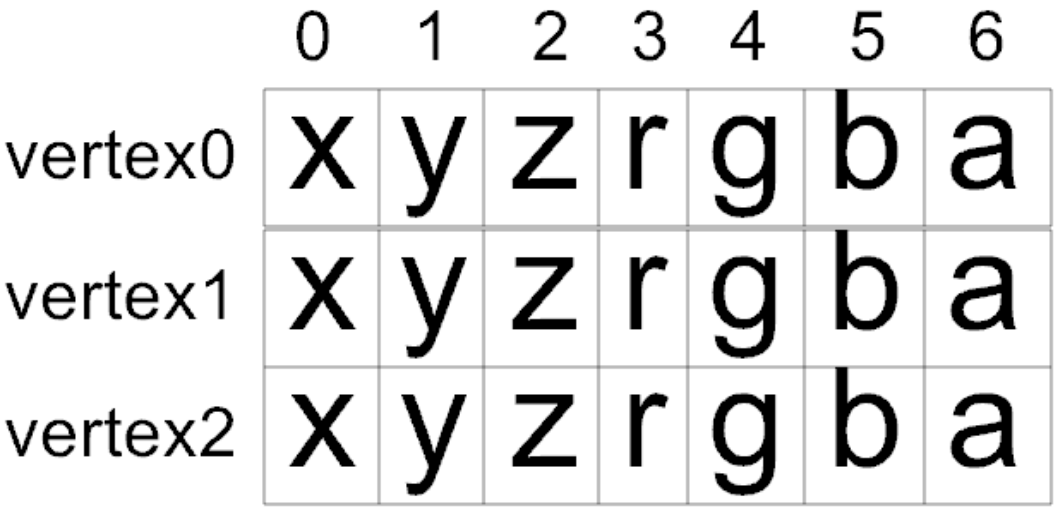


- Continuing from the previous step, change the shader to **eliminate the blue component** in the colour

Interleaved Vertex Attributes

KB-VIS
We Work In-Depth

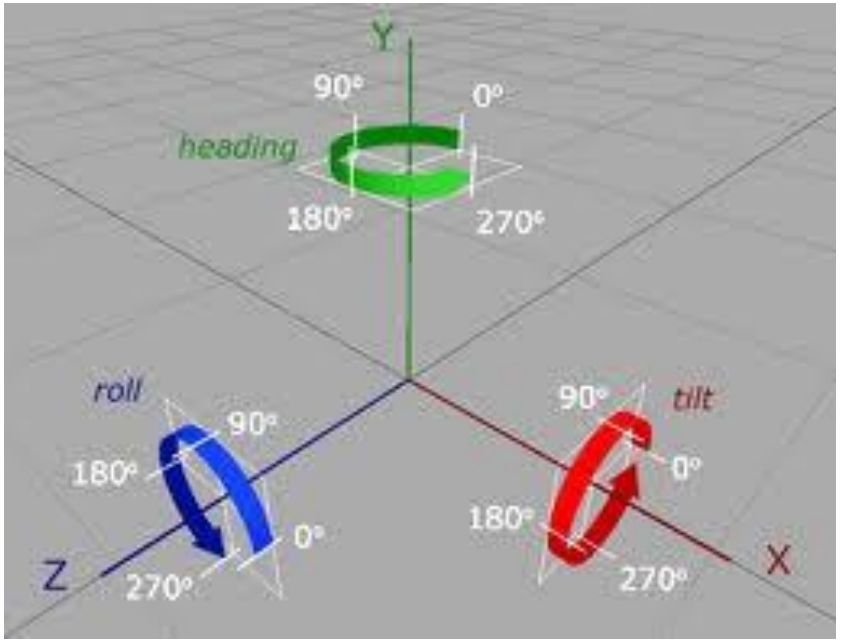
```
function initBuffers() {
    vertexAttributeBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, vertexAttributeBuffer);
    var vertexAndColorAttributes = [
        // triangle
        0.0, 1.0, 0.0,//XYZ
        1.0, 0.0, 0.0, 1.0,//RGBA
        -1.0, -1.0, 0.0,
        0.0, 1.0, 0.0, 1.0,
        1.0, -1.0, 0.0,
        0.0, 0.0, 1.0, 1.0,
        // square
        1.0, 1.0, 0.0,//XYZ
        0.5, 0.5, 1.0, 1.0,//RGBA
        -1.0, 1.0, 0.0,
        0.5, 0.5, 1.0, 1.0,
        1.0, -1.0, 0.0,
        0.5, 0.5, 1.0, 1.0,
        -1.0, -1.0, 0.0,
        0.5, 0.5, 1.0, 1.0,
    ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexAndColorAttributes), gl.STATIC_DRAW);
}
```



```
function drawScene() {
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0, pMatrix);
    mat4.identity(mvMatrix);
    mat4.translate(mvMatrix, [-1.5, 0.0, -7.0]);

    gl.bindBuffer(gl.ARRAY_BUFFER, vertexAttributeBuffer);
    var offset = 0;
    var stride = 28; // 7 floats (x,y,z and r,g,b,a)
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, 3, gl.FLOAT, false, stride, offset);
    // color values start 12 bytes after position (3 floats)
    var offset = 12;
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute, 4, gl.FLOAT, false, stride, offset);

    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLES, 0, 3);

    mat4.translate(mvMatrix, [3.0, 0.0, -1.0]);
    // use offset to skip first three entries of the triangle (7 floats per vertex)
    offset = 3 * 7 * 4;
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, 3, gl.FLOAT, false, stride, offset);
    // color values start 12 bytes after position (3 floats)
    offset += 12;
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute, 4, gl.FLOAT, false, stride, offset);

    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);

}
```

DAY 4

3D Rotation

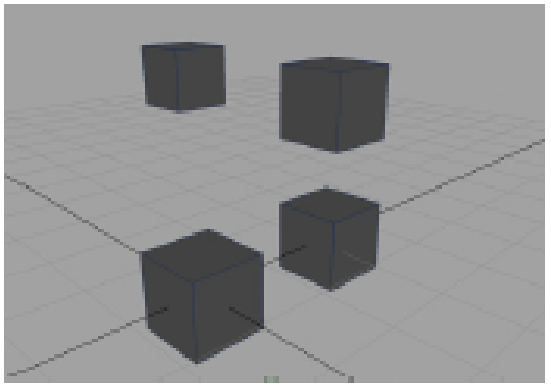


Right Hand Rule for Positive Direction of Rotation

Transformations

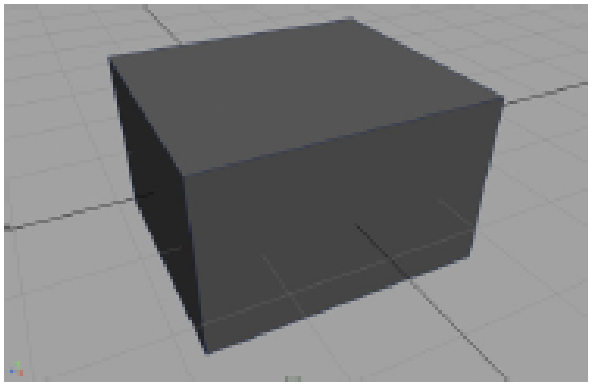
Translation

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



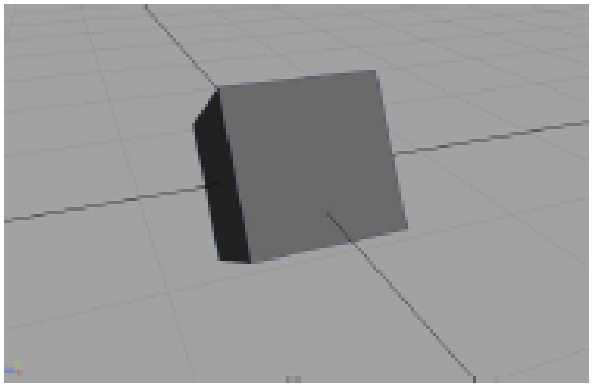
Scaling

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



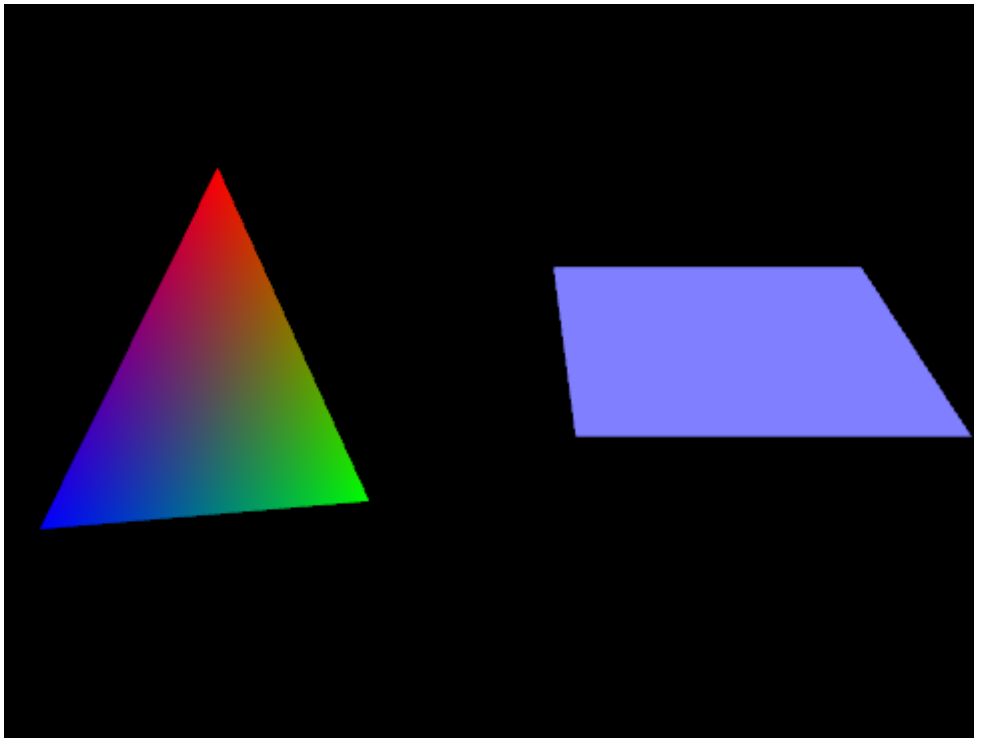
Rotation about z axis

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



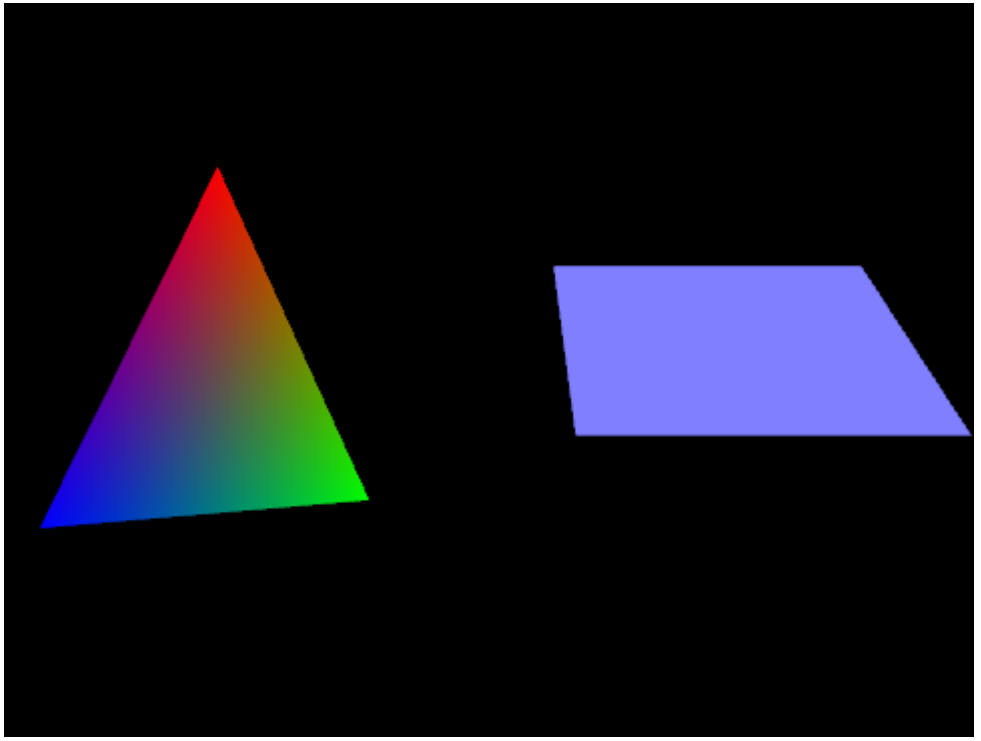
$$x' = T \cdot x$$

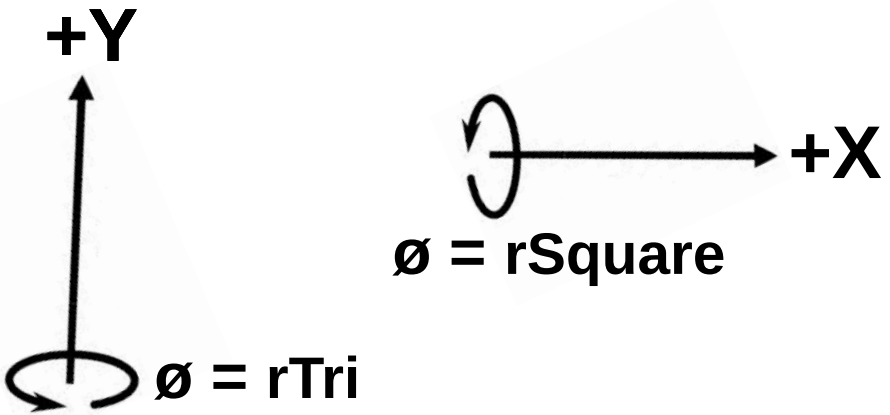
3D Rotation Example



See example03-rotation

Rotation Axes





The Matrix Transforms


```
function drawScene() {
            ...
      mat4.identity(mvMatrix);
      mat4.translate(mvMatrix, [-1.5, 0.0, -7.0]);
      // rotate the triangle
      mvPushMatrix();
      mat4.rotate(mvMatrix, degToRad(rTri), [0, 1, 0]);
      setMatrixUniforms();
      gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);
      // roll back to previous state
      mvPopMatrix();

      mat4.translate(mvMatrix, [3.0, 0.0, 0.0]);
      mvPushMatrix();
      // rotate the square
      mat4.rotate(mvMatrix, degToRad(rSquare), [1, 0, 0]);
      SetMatrixUniforms();
      ...
      gl.drawArrays(gl.TRIANGLE_STRIP, 0, squareVertexPositionBuffer.numItems);
      // roll back to previous state
      mvPopMatrix();
}
```

Animating the Geometry


```
function animate() {
    var timeNow = new Date().getTime();
    if (lastTime != 0) {
        var elapsed = timeNow - lastTime;

        rTri += (90 * elapsed) / 1000.0;
        rSquare += (75 * elapsed) / 1000.0;
    }
    lastTime = timeNow;
}

function tick() {
    requestAnimFrame(tick);
    drawScene();
    animate();
}

function webGLStart() {
    var canvas = document.getElementById("myCanvas");
    initGL(canvas);
    initShaders()
    initBuffers();

    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);

    tick();
}
```

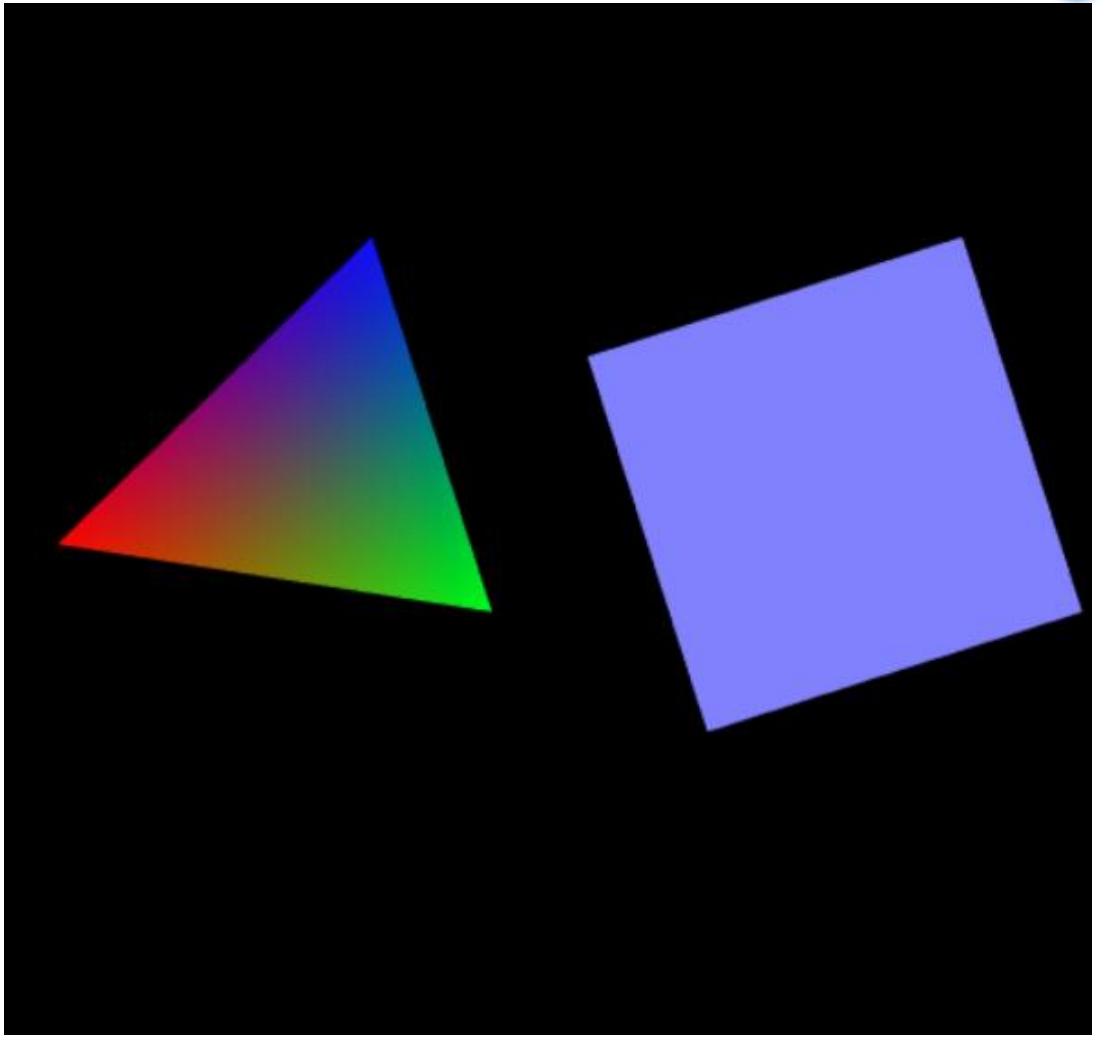
Animation Methods


```
handle = requestAnimationFrame( callback );
```

- Registers a function to call when the system is ready to update (repaint) the display
- Provides smoother animations and allows browsers to optimise better
- Recommended over traditional `setInterval`() and `setTimeout`() methods
- *`callback`* [in] - animation function to be called when system is ready
- *`handle`* [out, retval] - handle to the requestAnimationFrame method that can be used to cancel the request if needed

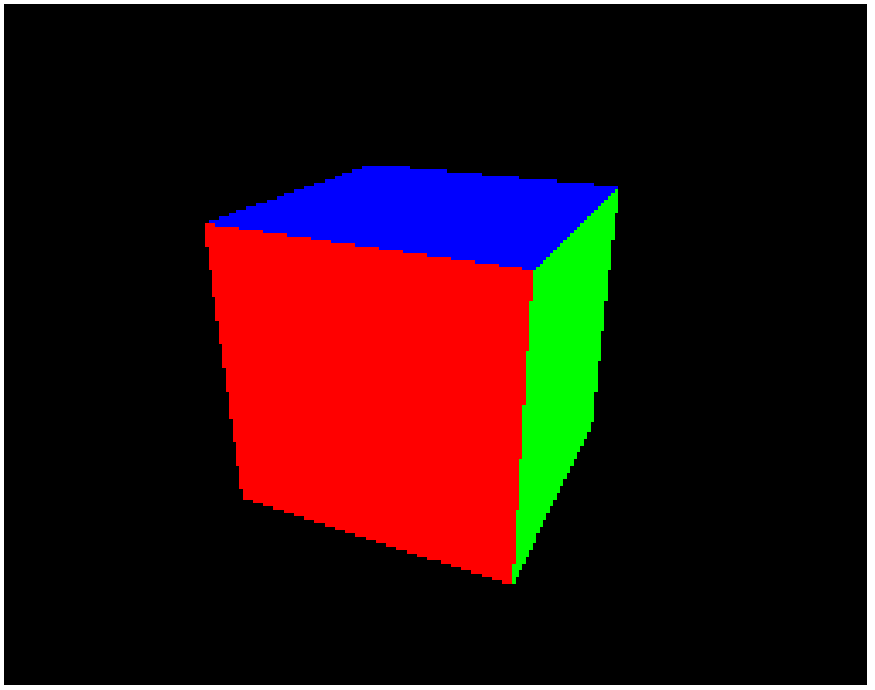
`requestAnimFrame() -` wrapper from *webgl-utils.js* (JavaScript utilities for common WebGL tasks)

Exercise 3.1



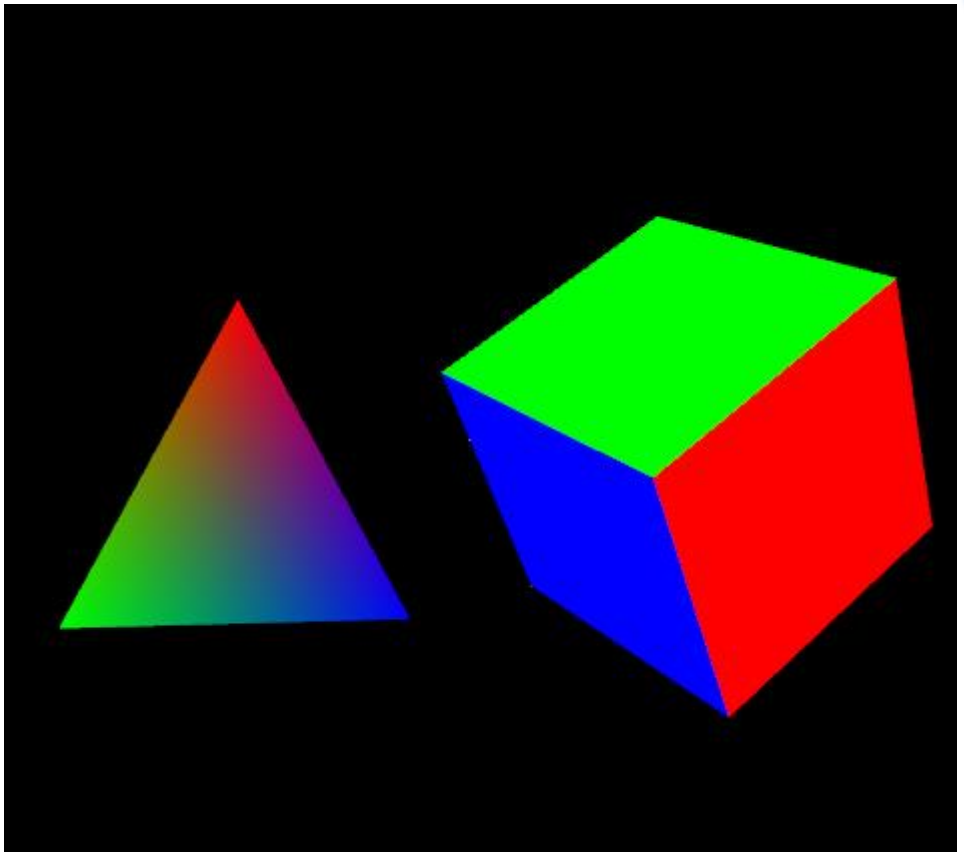
- Change the Axis of Rotation to Z-Axis
- Mail your index.html to 3d@kbvis.com

Moving on to 3D...



See examples/coloring folder

3D Objects

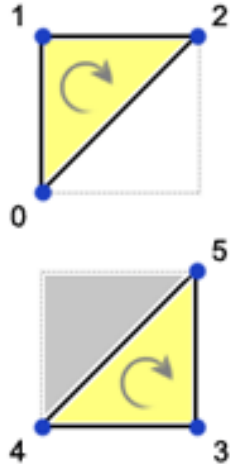


See examples/3d-shapes

Vertex and Index Buffers

Vertex Buffer

VB Index :	
0	(-1, -1)
1	(-1, 1)
2	(1, 1)
3	(1, -1)
4	(-1, -1)
5	(1, 1)

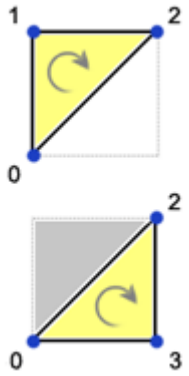


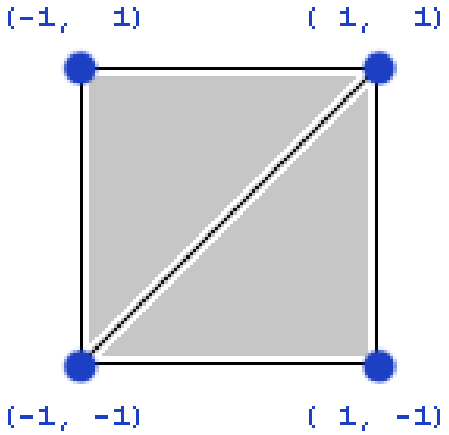
Index Buffer

IB Index :	
0	0
1	1
2	2
3	3
4	0
5	2

Vertex Buffer

VB Index :	
0	(-1, -1)
1	(-1, 1)
2	(1, 1)
3	(1, -1)





with Indexing

Using Vertex and Index Buffers to Define a Cube


```
function initBuffers() {
       cubeVertexPositionBuffer = gl.createBuffer();
       gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
       vertices = [
           // Front face
           -1.0, -1.0,  1.0,
            1.0, -1.0,  1.0,
            1.0,  1.0,  1.0,
           -1.0,  1.0,  1.0,

           // Back face
           -1.0, -1.0, -1.0,
           -1.0,  1.0, -1.0,
            1.0,  1.0, -1.0,
            1.0, -1.0, -1.0,

           // Top face
           -1.0,  1.0, -1.0,
           -1.0,  1.0,  1.0,
            1.0,  1.0,  1.0,
            1.0,  1.0, -1.0,

           // Bottom face
           -1.0, -1.0, -1.0,
            1.0, -1.0, -1.0,
            1.0, -1.0,  1.0,
           -1.0, -1.0,  1.0,

           // Right face
            1.0, -1.0, -1.0,
            1.0,  1.0, -1.0,
            1.0,  1.0,  1.0,
            1.0, -1.0,  1.0,

           // Left face
           -1.0, -1.0, -1.0,
           -1.0, -1.0,  1.0,
           -1.0,  1.0,  1.0,
           -1.0,  1.0, -1.0
       ];
       gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
       cubeVertexPositionBuffer.itemSize = 3;
       cubeVertexPositionBuffer.numItems = 24;
```

- Specify cube vertices
- Vertices are not shared across faces, as colours are different

```
       cubeVertexColorBuffer = gl.createBuffer();
       gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexColorBuffer);
       colors = [
           [1.0, 0.0, 0.0, 1.0], // Front face
           [1.0, 1.0, 0.0, 1.0], // Back face
           [0.0, 1.0, 0.0, 1.0], // Top face
           [1.0, 0.5, 0.5, 1.0], // Bottom face
           [1.0, 0.0, 1.0, 1.0], // Right face
           [0.0, 0.0, 1.0, 1.0]  // Left face
       ];
       var unpackedColors = [];
       for (var i in colors) {
           var color = colors[i];
           for (var j=0; j < 4; j++) {
               unpackedColors = unpackedColors.concat(color);
           }
       }
       gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(unpackedColors),
                     gl.STATIC_DRAW);
       cubeVertexColorBuffer.itemSize = 4;
       cubeVertexColorBuffer.numItems = 24;

       cubeVertexIndexBuffer = gl.createBuffer();
       gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
       var cubeVertexIndices = [
           0, 1, 2,      0, 2, 3,    // Front face
           4, 5, 6,      4, 6, 7,    // Back face
           8, 9, 10,     8, 10, 11,  // Top face
           12, 13, 14,   12, 14, 15, // Bottom face
           16, 17, 18,   16, 18, 19, // Right face
           20, 21, 22,   20, 22, 23  // Left face
       ];
       gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new
                                      Uint16Array(cubeVertexIndices),
gl.STATIC_DRAW);
       cubeVertexIndexBuffer.itemSize = 1;
       cubeVertexIndexBuffer.numItems = 36;
}
```

- Define cube geometry by indexing into vertex buffer
- Index of shared vertex repeated, not the (x,y,z) position

KB-VIS
We Work In-Depth

DAY 5

OpenGL Shading Language (GLSL)

- GLSL is very similar to C
- Source code sent to GPU as a string
- Shader gets compiled and linked
- GLSL Types:

Type	Description
void	No *function return* value or *empty parameter* list.
float	Floating point value.
int	Signed integer.
bool	Boolean value.
vec2, vec3, vec4	Floating point vector.
ivec2, ivec3, ivec4	Signed integer vector.
bvec2, bvec3, bvec4	Boolean vector.
mat2, mat3, mat4	2x2, 3x3, 4x4 floating point matrix.
sampler2D	Access a 2D texture.
samplerCube	Access cube mapped texture.

Before GLSL


```
void KB_VolumeRenderer::CreateAtiShader()
{
   m_unShader3D = glGenFragmentShadersATI(1);
   glBindFragmentShaderATI(m_unShader3D);

   glBeginFragmentShaderATI();

   glSampleMapATI(GL_REG_0_ATI, GL_TEXTURE0_ARB, GL_SWIZZLE_STR_ATI);
   glColorFragmentOp1ATI(GL_MOV_ATI, GL_REG_1_ATI, GL_NONE, GL_NONE, GL_REG_0_ATI,
            GL_NONE, GL_NONE);

   glSampleMapATI(GL_REG_1_ATI, GL_REG_1_ATI, GL_SWIZZLE_STR_ATI);
   glColorFragmentOp1ATI(GL_MOV_ATI, GL_REG_0_ATI, GL_NONE, GL_NONE, GL_REG_1_ATI,
            GL_NONE, GL_NONE);

   glEndFragmentShaderATI();
}
```

UGLY! – like writing assembly code

OpenGL and GLSL

OpenGL Version	GLSL Version
2.0	1.10
2.1	1.20
3.0	1.30
3.1	1.40
3.2	1.50

In OpenGL 3.3+ version number is the same

https://www.khronos.org/registry/OpenGL/specs/gl/

GLSL Qualifiers

- ***in* –** input to the shader stage
- ***attribute*** - linkage between a vertex shader and OpenGL for per-vertex data. Deprecated, now same as ***in*** variables.
- ***uniform*** - linkage between a shader and OpenGL for per-render data
- ***varying*** - linkage between the vertex shader and the fragment shader for interpolated data. Deprecated, now same as ***in/out*** variables.
- ***out*** variable of the vertex shader needs to match ***in*** variable of the fragment shader
- **Built-in Variables:**
  - vec4 **gl_Position** - vertex shader needs to set this variable to the computed position
  - vec4 **gl_FragColor** - defines fragment's RGBA color that will eventually be written to the frame buffer, set by the fragment shader. Post GLSL 130 output fragments may be user-specified

GLSL Uniforms

Uniform variables must be defined in GLSL at global scope.

Uniforms can be of any type, or any aggregation of types. The following is all legal GLSL code:

```
struct TheStruct
{
  vec3 first;
  vec4 second;
  mat4x3 third;
};

uniform vec3 oneUniform;
uniform TheStruct aUniformOfArrayType;
uniform mat4 matrixArrayUniform[25];
uniform TheStruct uniformArrayOfStructs[10];
```

https://www.khronos.org/opengl/wiki/Uniform_(GLSL)

Migrating to WebGL2

- Call canvas.getContext("**webgl2**") instead of “webgl1”
- No “experimental” context anymore
- Switch to GLSL 300 es
- Add “**#version 300 es**” **right at the beginning** of your shaders
- Replace “**attribute**” with “**in**”
- Replace “**varying**” with “**in**” or “**out**”
- Replace “**gl_FragColor**” with your own output color

https://www.khronos.org/opengl/wiki/OpenGL_Shading_Language

Example 04 – WebGL2 Version

KB-VIS
We Work In-Depth

```
function initGL(canvas) {
    try {
        gl = canvas.getContext("webgl2");
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    } catch (e) {
    }
    if (!gl) {
        alert("Could not initialise WebGL, sorry :-(");
    }
}
```

Example 04 – WebGL2 Version

KB-VIS
We Work In-Depth

```
<script id="shader-fs" type="x-shader/x-fragment">#version 300 es

    precision mediump float;

    in vec4 vColor;

    out vec4 fragColor;

    void main(void) {
        fragColor = vColor;
    }
</script>

<script id="shader-vs" type="x-shader/x-vertex">#version 300 es

    in vec3 aVertexPosition;
    in vec4 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    out vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = aVertexColor;
    }
</script>
```

Interpolation qualifiers

Interpolation qualifiers control how interpolation of values happens across a triangle or other primitive.

- **flat**

The value will not be interpolated. The value given to the fragment shader is the value from the Provoking Vertex for that primitive.

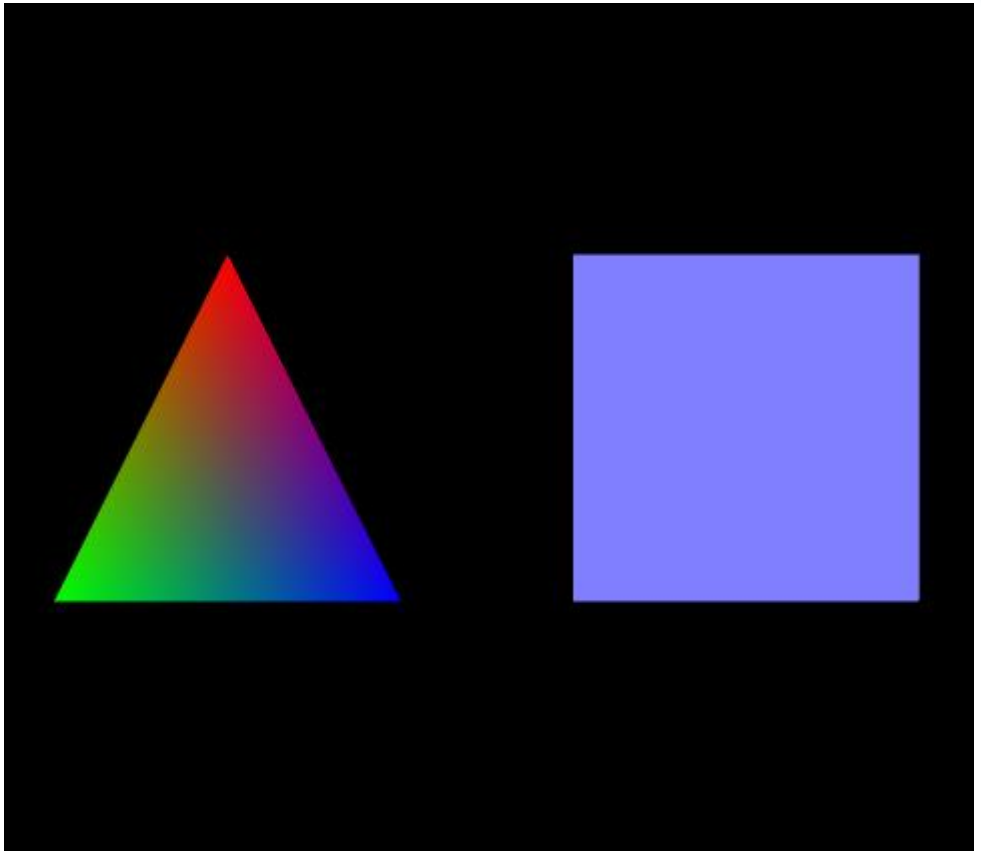
- **noperspective**

The value will be linearly interpolated in window-space. This is usually not what you want, but it can have its uses.

- **smooth**

The value will be interpolated in a perspective-correct fashion. This is the default if no qualifier is present.

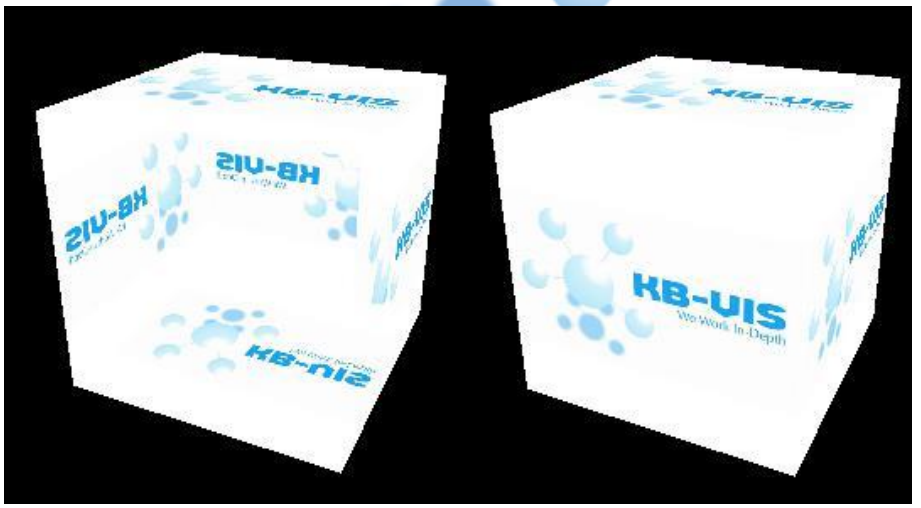
Flat Colour

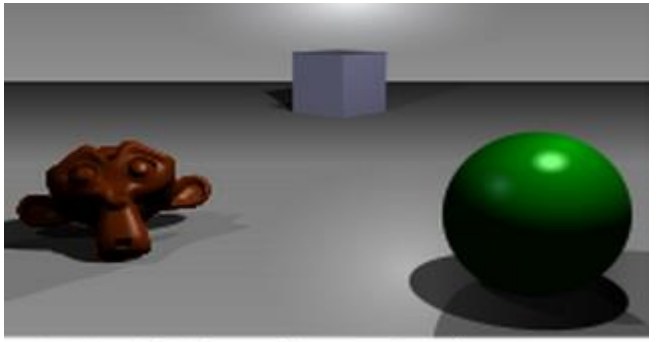


See example02-colour/flat-interpolation-webgl2.html

Depth Buffering

- Brute force hardware solution to visibility ordering
- Depth buffer stores distance between the viewpoint and the object occupying that pixel
- if the incoming fragment is nearer, the incoming depth value replaces the one already in the depth buffer
- **gl.depthFunc**() - sets the comparison function for the depth test (default is gl.LESS)



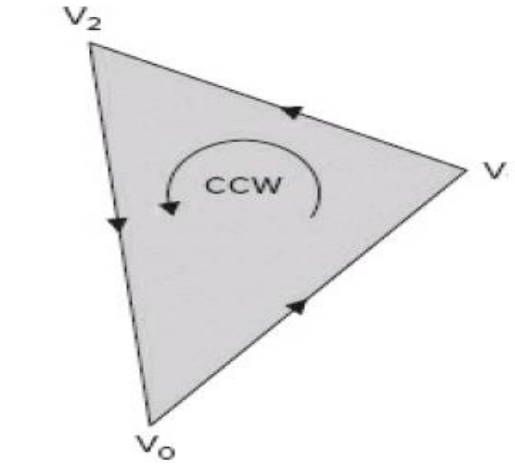


A simple three-dimensional scene

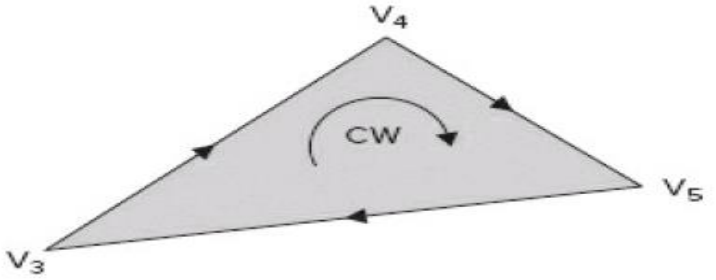
Z-buffer representation

Culling

- **gl.cullFace**(mode) - indicates which triangles should be discarded (culled)
- mode is either gl.FRONT, gl.BACK, or gl.FRONT_AND_BACK
- culling must be enabled using **gl.enable**(gl.CULL_FACE)
- triangles whose vertices appear in counterclockwise order on the screen are called front-facing

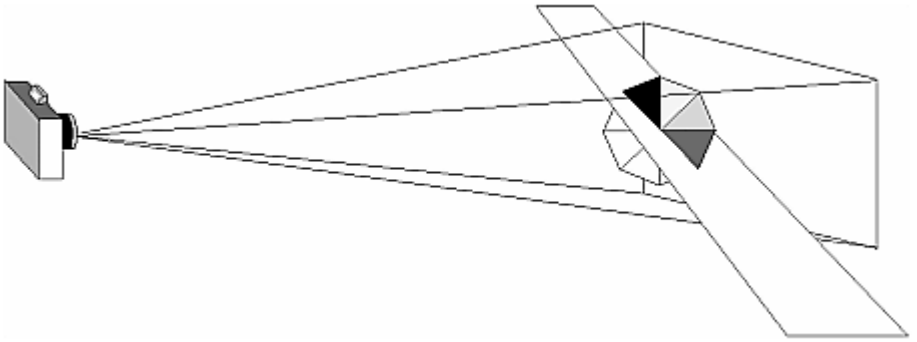


Counterclockwise Winding



Clockwise Winding

Clipping



Plane Equation:

Ax + **B**y + **C**z + **D** = 0

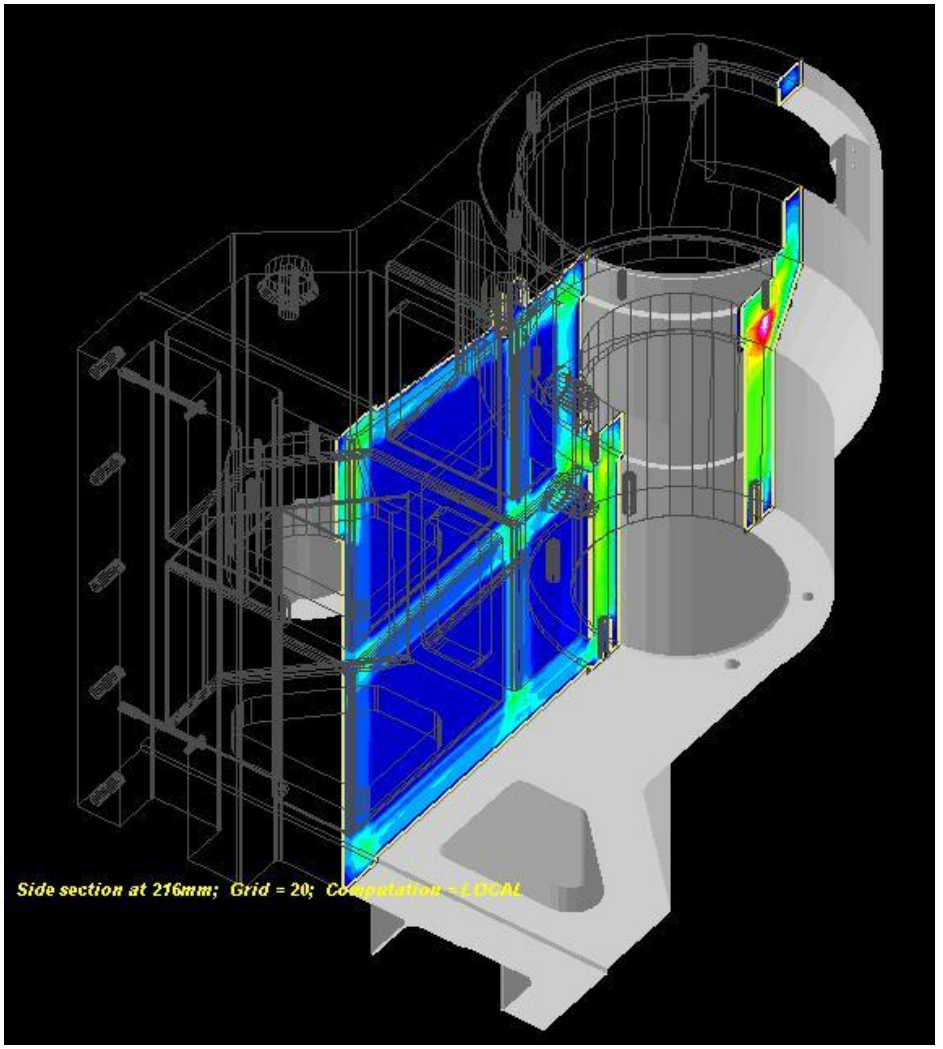
(A,B,C) = plane normal vector

D = Distance from origin

Cutaway View

KB-VIS
We Work In-Depth

- Render object with clipplane enabled
- Disable clipping, render wireframe, with depth-test enabled



Fixed Function API:

```
void glClipPlane(plane, equation);
```

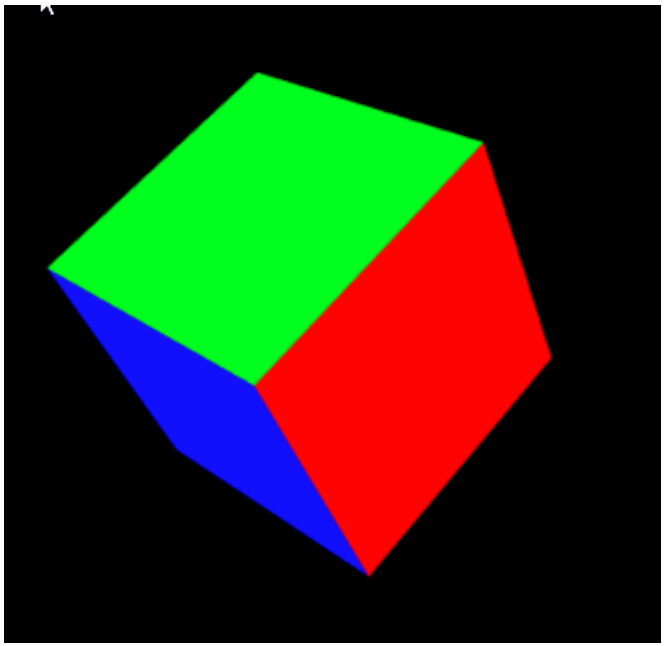
Example:

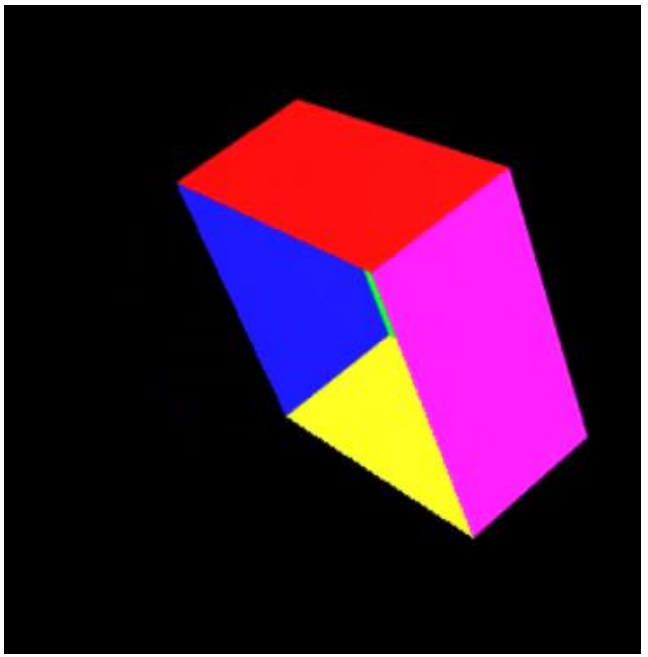
```
    eqn[] = {a, b, c, d};
    glClipPlane (GL_CLIP_PLANE0, eqn);
    glEnable (GL_CLIP_PLANE0);
```

Programmable API:

```
in vec4 Position;
out float gl_ClipDistance[1];

uniform mat4 Projection;
uniform mat4 Modelview;
uniform mat4 ViewMatrix;
uniform mat4 ModelMatrix;
uniform vec4 ClipPlane;

void main()
{
    vPosition = Position.xyz;
    gl_Position = Projection * Modelview * Position;
    gl_ClipDistance[0] = dot(ModelMatrix * Position,
          ClipPlane);
}
```

Example 04
(Clipping Added)

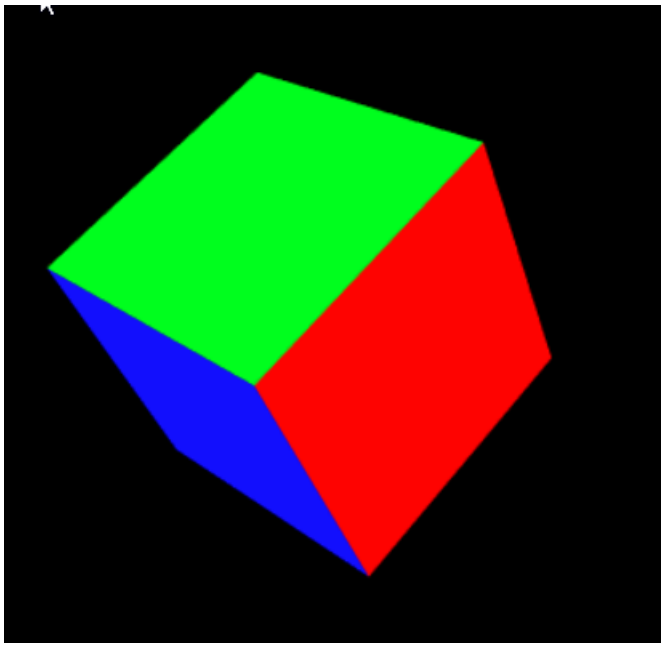


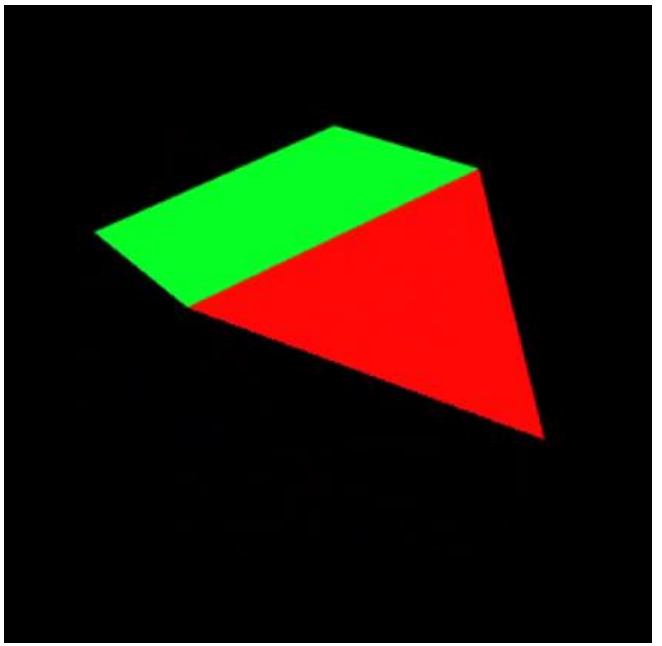


Clip against
XZ Plane (Y=0)

```
<script id="shader-vs" type="x-shader/x-vertex">#version 300 es // has to be on first line - do not add newline

    in vec3 aVertexPosition;
    in vec4 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    out vec4 vColor;
    out vec3 untransformedVertex;

    void main(void) {
        untransformedVertex = aVertexPosition;
        vColor = aVertexColor;
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
    }
</script>

<script id="shader-fs" type="x-shader/x-fragment">#version 300 es // has to be on first line - do not add newline

    precision mediump float;

    in vec4 vColor;
    in vec3 untransformedVertex;

    out vec4 fragColor;

    void main(void) {
        // if clipped by z=1 plane, discard fragment
        if(dot(untransformedVertex, vec3(0.0,1.0,0.0)) < 0.0) discard;
        fragColor = vColor;
    }
</script>
```

Exercise 4.1

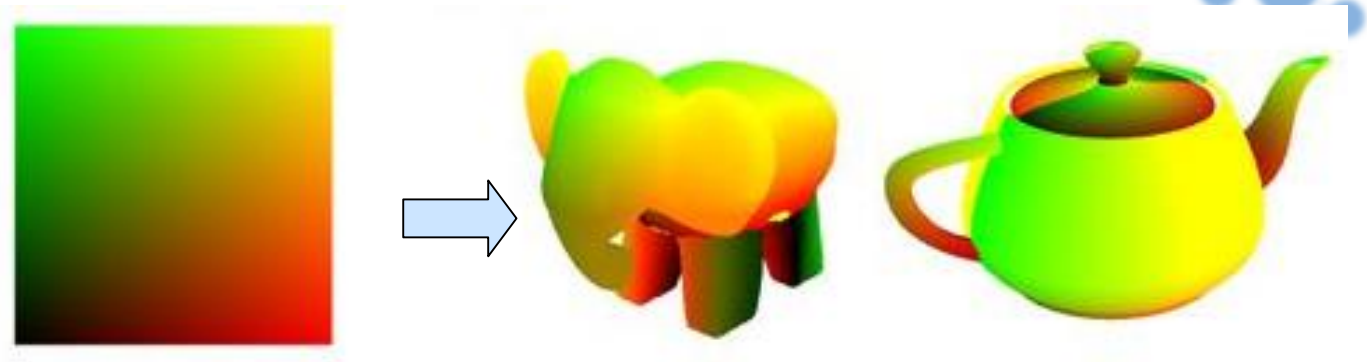


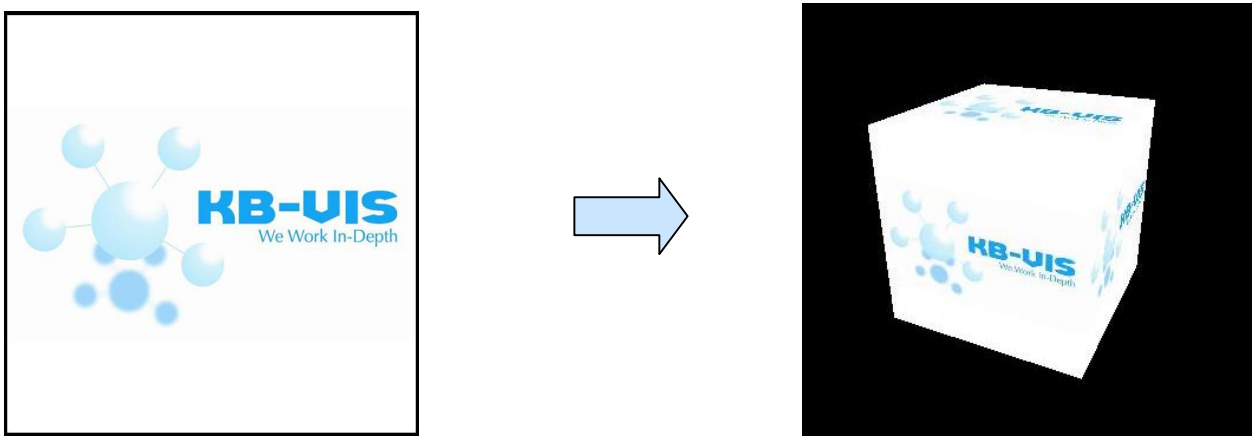


Clip Cube to Prism

- Mail your index.html to 3d@kbvis.com
- Feel free to add any questions

Texture Mapping

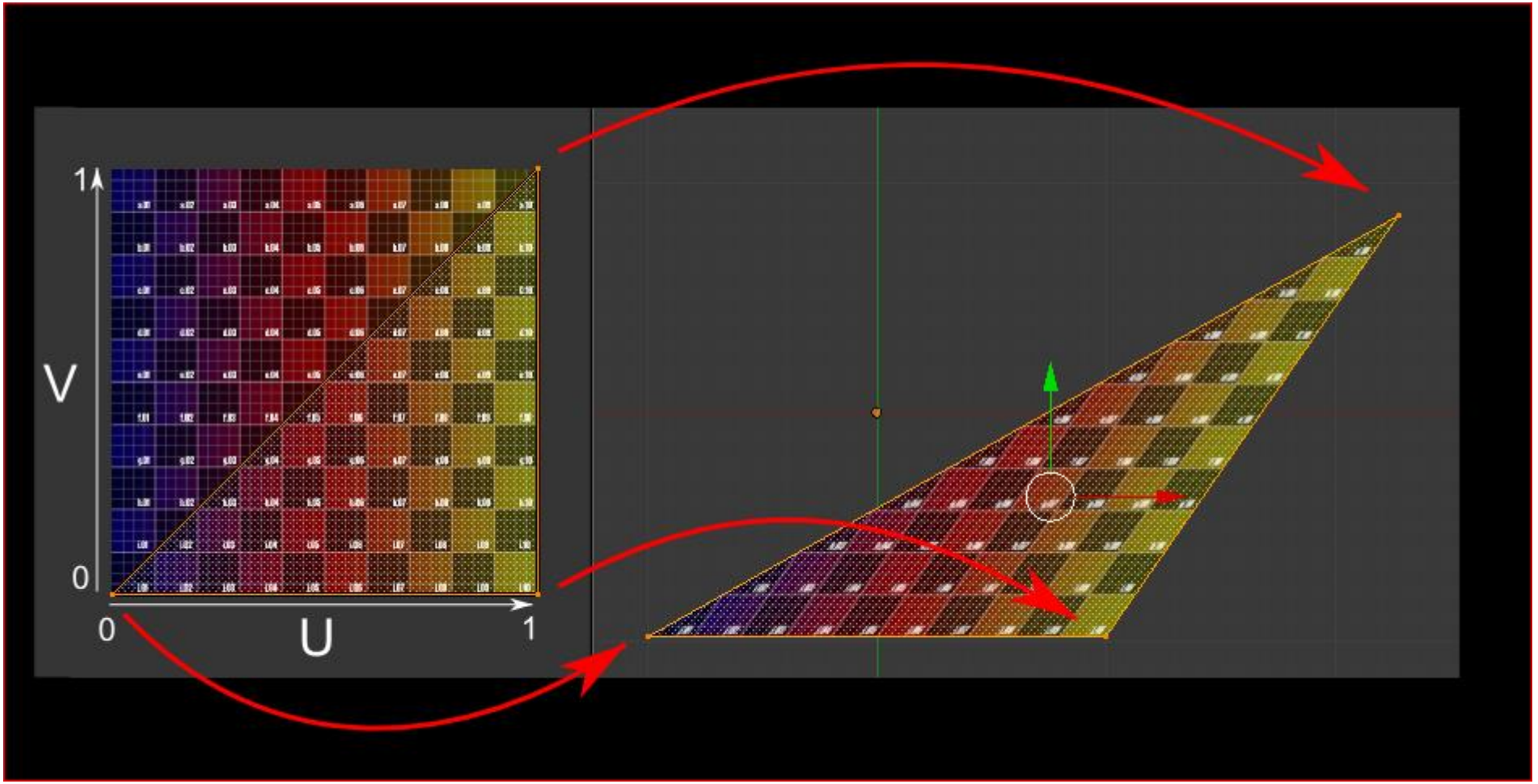




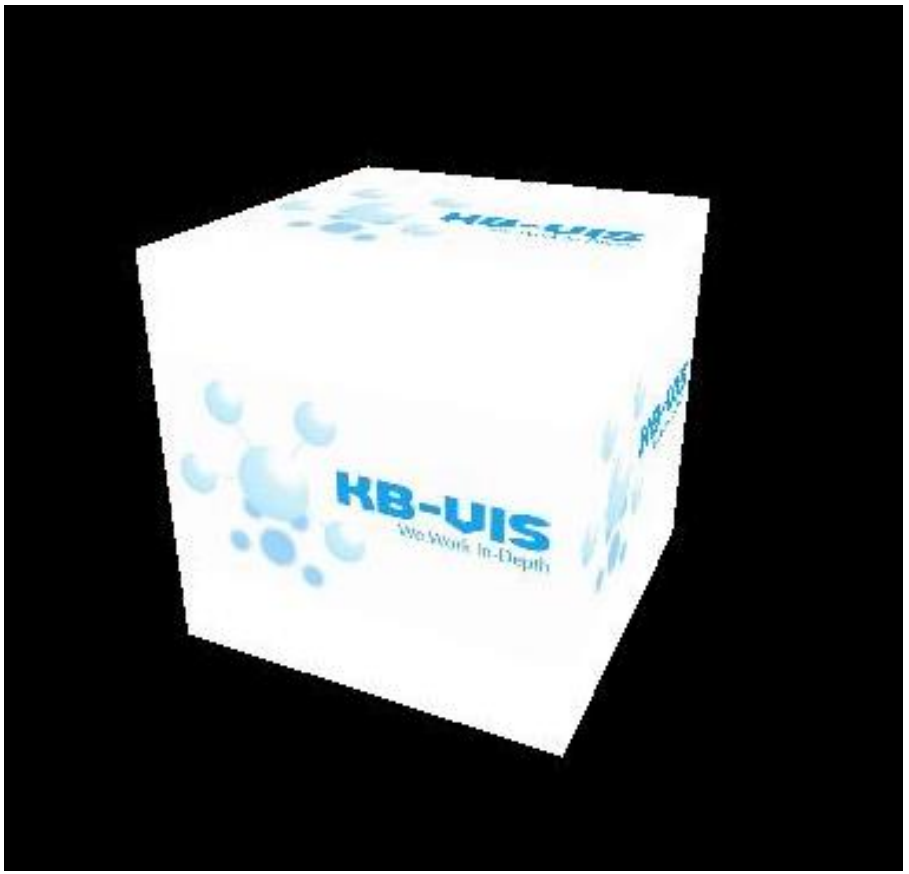
- Application of images onto the model surface
- Texture Coordinates – [0, 1.0]
- Values < 0.0 or > 1.0 are clamped or wrapped
- Multiple textures may be used

(u,v) – (x,y,z) Mapping

KB-VIS
We Work In-Depth

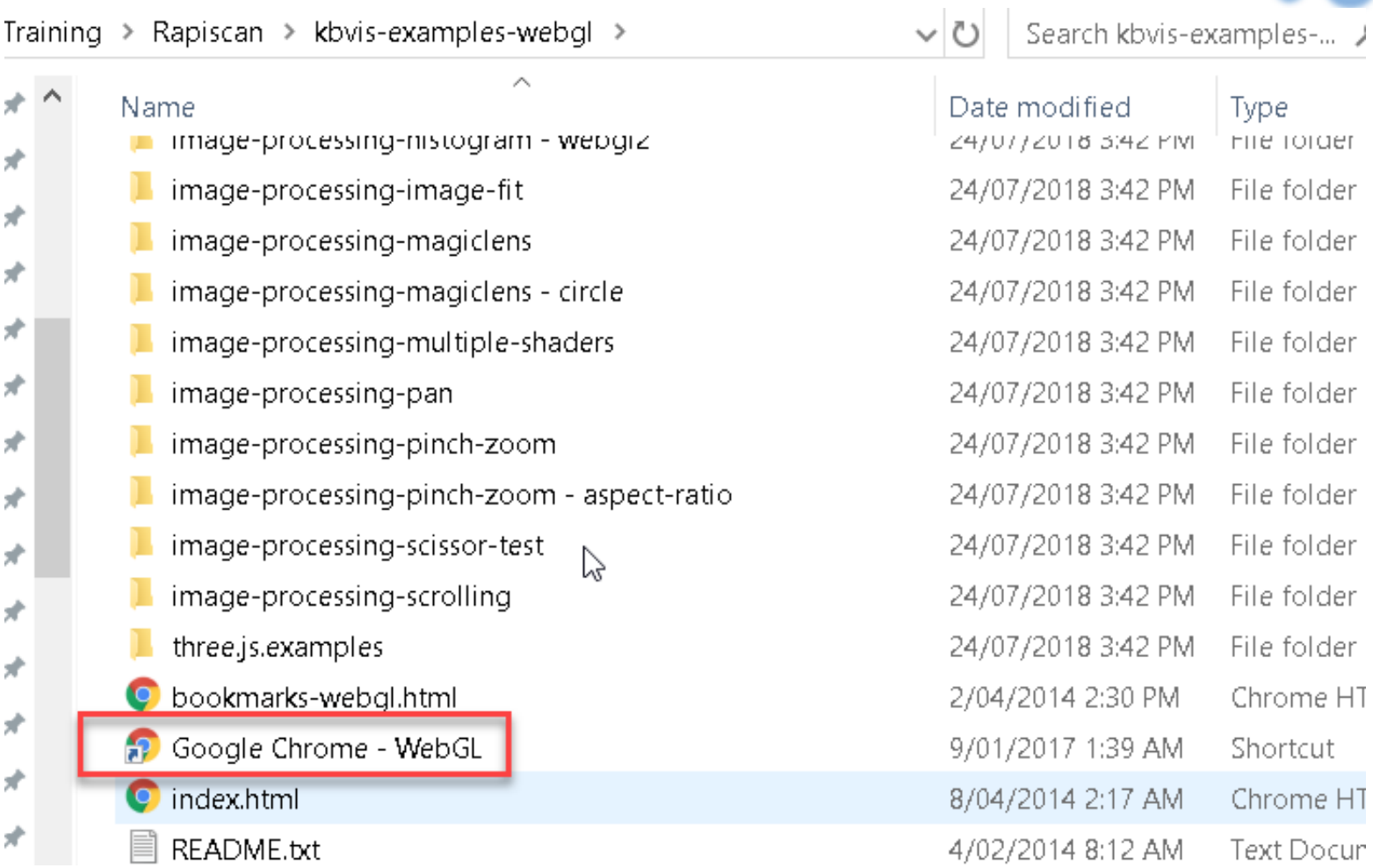


Textured Cube



See examples/texture

Chrome – Enable File Access



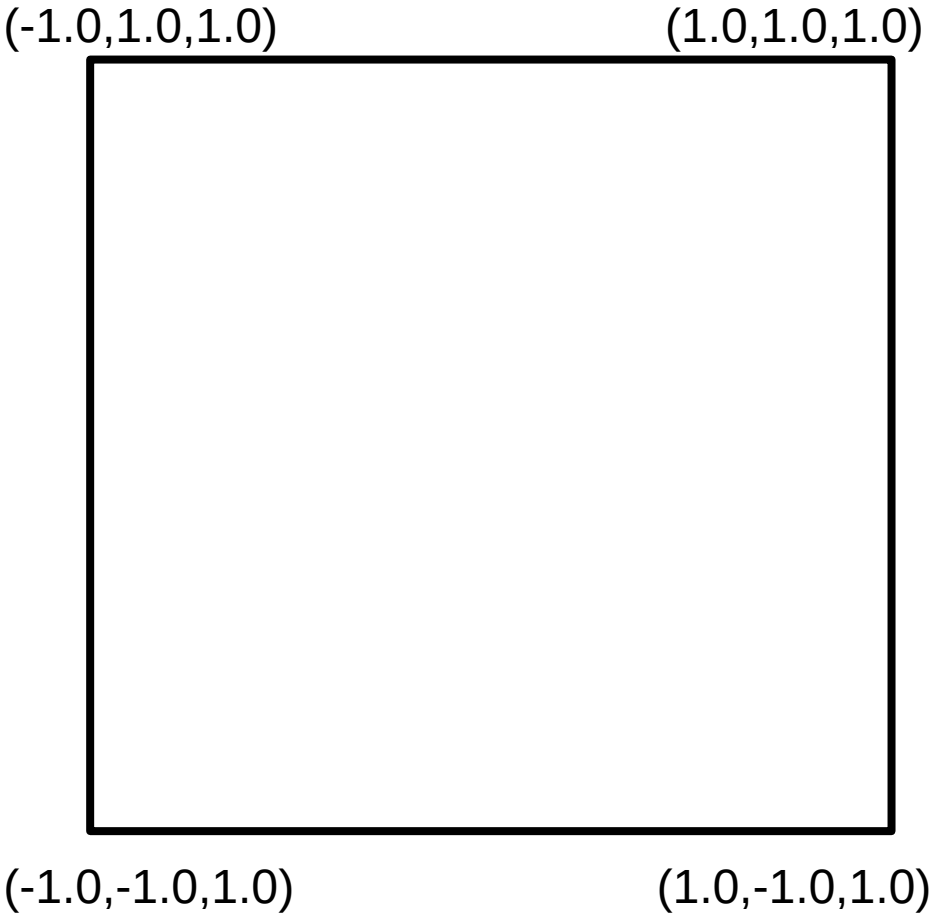
"C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --allow-file-access-from-files --enable-unsafe-es3-apis

Defining Texture Attributes


```
function initBuffers() {
      ...
    cubeVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    var textureCoords = [
      // Front face
      0.0, 0.0,
      1.0, 0.0,
      1.0, 1.0,
      0.0, 1.0,

      // Back face
      1.0, 0.0,
      1.0, 1.0,
      0.0, 1.0,
      0.0, 0.0,
              ...
    ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoords), gl.STATIC_DRAW);
    cubeVertexTextureCoordBuffer.itemSize = 2;
    cubeVertexTextureCoordBuffer.numItems = 24;
      ...
}
```

- Allocate and populate the texture coordinates vertex attribute buffer using `bufferData()`

Texture Coordinates



Vertex Coordinates

Texture Coordinates

Loading the Texture


```
function handleLoadedTexture(texture) {
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, texture.image);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
    gl.bindTexture(gl.TEXTURE_2D, null);
}
```

```
void texImage2D(GLenum target, GLint level, GLenum
                        internalformat, GLenum format, GLenum type,
HTMLImageElement image)
```

used to load texture data

```
void texParameter(GLenum target, GLenum pname, GLint param)
```

used to set texture mode (repeat / clamp, filtering mode – linear / nearest)

Shaders (WebGL 1)


```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec2 aTextureCoord;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying vec2 vTextureCoord;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vTextureCoord = aTextureCoord;
    }
</script>

<script id="shader-fs" type="x-shader/x-fragment">
    precision mediump float;

    varying vec2 vTextureCoord;

    uniform sampler2D uSampler;

    void main(void) {
        gl_FragColor = texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
    }
</script>
```

- The vertex attribute (`aTextureCoord`) is passed to the fragment shader as a varying (`vTextureCoord`) to be interploated across fragments
- The fragment shader looks up the texture map (`uniform sampler2D uSampler`) using the interpolated texture coordinate in `vTextureCoord`

Shaders (WebGL 2)


```
<script id="shader-vs" type="x-shader/x-vertex">#version 300 es
    in vec3 aVertexPosition;
    in vec2 aTextureCoord;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    out vec2 vTextureCoord;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vTextureCoord = aTextureCoord;
    }
</script>

<script id="shader-fs" type="x-shader/x-fragment">#version 300 es
    precision mediump float;

    uniform sampler2D uSampler;

    in vec2 vTextureCoord;

    out vec4 fragColor;

    void main(void) {
        fragColor = texture(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
    }
</script>
```

- The vertex attribute (`aTextureCoord`) is passed to the fragment shader as output (`vTextureCoord`) to be interploated across fragments
- The fragment shader looks up the texture map (`uniform sampler2D uSampler`) using the interpolated texture coordinate in `vTextureCoord`

Enabling Texture Attributes for Drawing


```
function drawScene() {
    ...
    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
                           cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
                   cubeVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, myTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
}
```

- `activeTexture()` — select active texture unit (you can have multiple texture units active)
- Use `gl.uniform1i()` to set `samplerUniform` to the index of the texture unit (0) you want the fragment shader to use

KB-VIS
We Work In-Depth

DAY 6

OpenGL Texture API

- glCreateTextures / glGenTextures
- glBindTexture
- glActiveTexture
- glTexImage(1D/2D/3D)
- glTexSubImage(1D/2D/3D)
- glTexParameter
- glDeleteTextures

See http://docs.gl/es2/

C Specification

```
void glCreateTextures( GLenum target,
                       GLsizei n,
                       GLuint *textures);
```

Parameters

target

Specifies the effective texture target of each created texture.

n

Number of texture objects to create.

textures

Specifies an array in which names of the new texture objects are stored.

- WebGL: **gl.createTexture()** returns the texture name

glActiveTexture

glActiveTexture — select active texture unit

C Specification

```
void glActiveTexture(GLenum texture);
```

Parameters

texture

Specifies which texture unit to make active. The number of texture units is implementation dependent, but must be at least 8. *texture* must be one of GL_TEXTURE*i*, where i ranges from 0 to (GL_MAX_COMBINED_TEXTURE_IMAGE_UNITS - 1). The initial value is GL_TEXTURE0.

Description

`glActiveTexture` selects which texture unit subsequent texture state calls will affect. The number of texture units an implementation supports is implementation dependent, but must be at least 8.

glBindTexture

OpenGL 4
OpenGL 3
OpenGL 2
OpenGL ES 3
OpenGL ES 2

glBindTexture — bind a named texture to a texturing target

C Specification

```
void glBindTexture(GLenum target,
                   GLuint texture);
```

Parameters

target

Specifies the target of the active texture unit to which the texture is bound. Must be either `GL_TEXTURE_2D` or `GL_TEXTURE_CUBE_MAP`.

texture

Specifies the name of a texture.

glTexImage2D

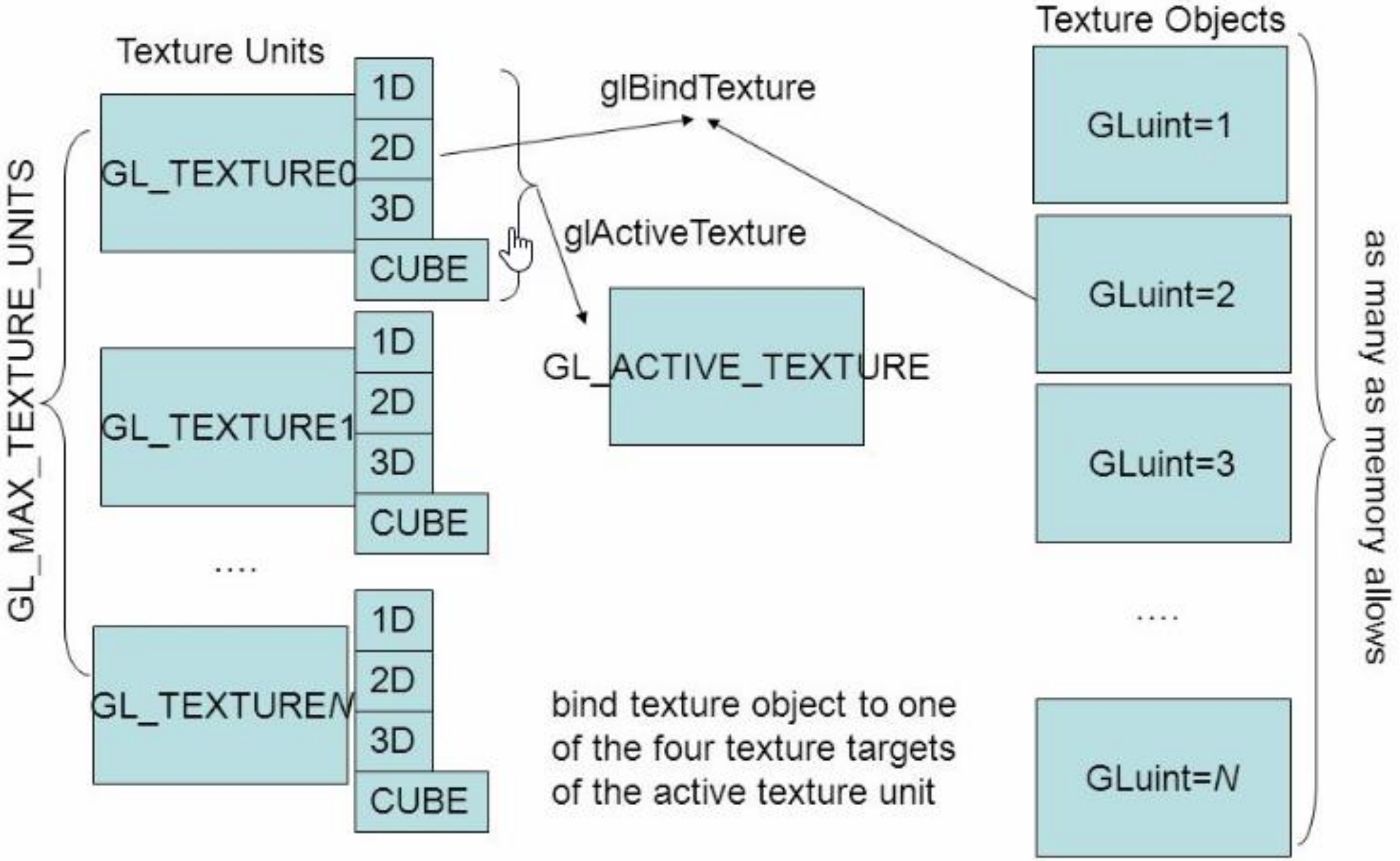
glTexImage2D — specify a two-dimensional texture image

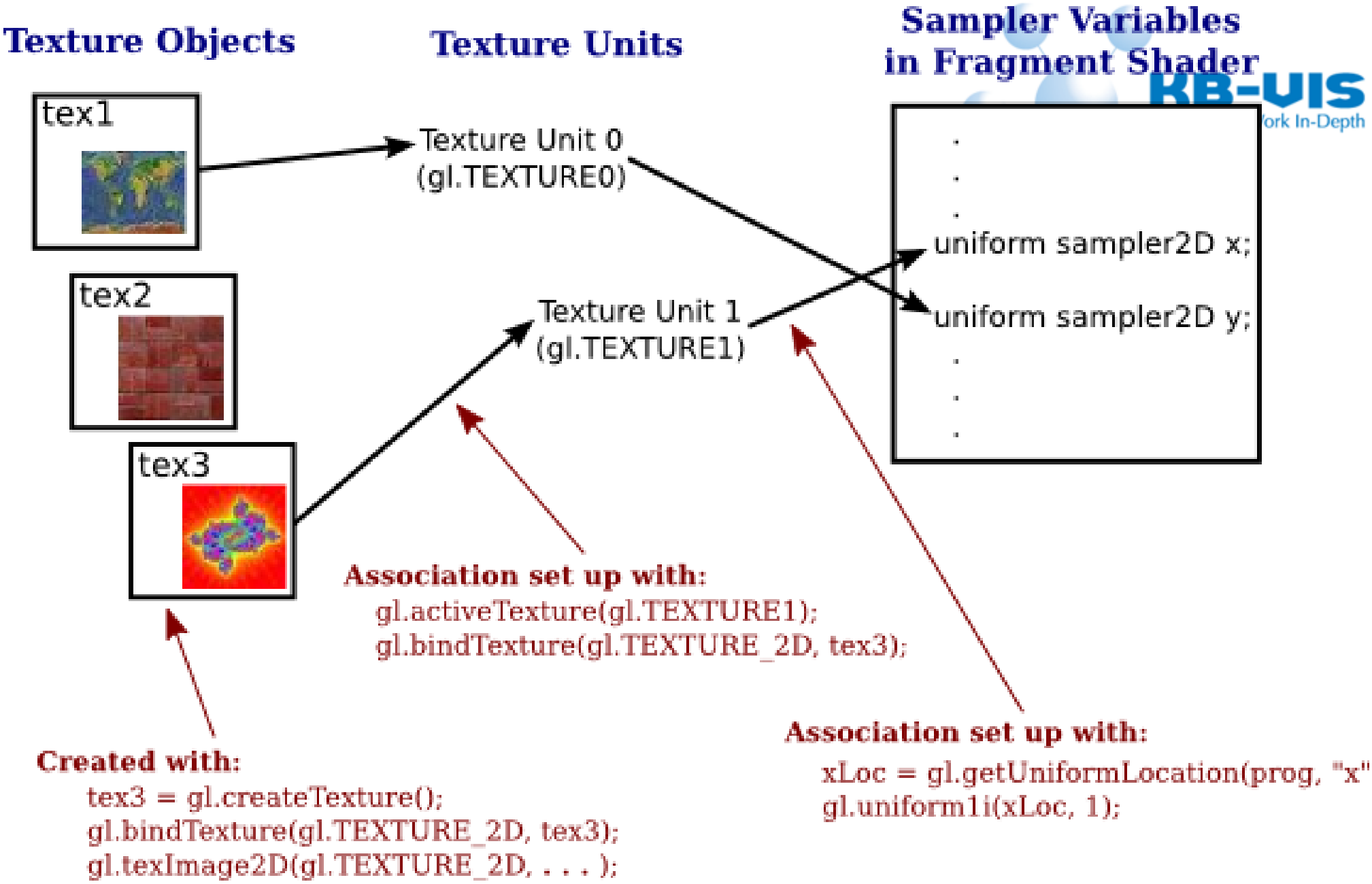
C Specification

```
void glTexImage2D(GLenum target,
                  GLint level,
                  GLint internalformat,
                  GLsizei width,
                  GLsizei height,
                  GLint border,
                  GLenum format,
                  GLenum type,
                  const GLvoid * data);
```

e.g. **gl.texImage2D**(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, texture.image);

Texture Unit → glBindTexture←Texture Object




Texture Objects
Texture Units
Sampler Variables
in Fragment Shader
tex1
tex2
tex3
Texture Unit 0
(gl.TEXTURE0)
Texture Unit 1
(gl.TEXTURE1)
uniform sampler2D x;
uniform sampler2D y;
Association set up with:
gl.activeTexture(gl.TEXTURE1);
gl.bindTexture(gl.TEXTURE_2D, tex3);
Created with:
tex3 = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, tex3);
gl.texImage2D(gl.TEXTURE_2D, . . .);
Association set up with:
xLoc = gl.getUniformLocation(prog, "x");
gl.uniform1i(xLoc, 1);

Confused?

- **Textures** can be many in number
  - e.g. we might have hundreds of bitmap images to choose from
  - Each texture is identified by its name (unsigned integer)
- **Texture Units** are limited (only 8 guaranteed)
- Texture Unit (TU/TMU/TPU) - hardware component in a GPU that does sampling
- A Texture Unit can only be bound to a single texture
- A Texture can be bound to multiple texture units
- **glActiveTexture** specifies which TU we are dealing with
- **glBindTexture** specifies which Texture Map we are dealing with

Texturing in the Shader

- **Sampler Uniform**: used to perform texture lookup in the shader
- Type *1D/2D/3D/Cube/Shadow* (let's stick to 2D for now)
- e.g. `uniform sampler2D uMyTextureMap;`
- Texture is sampled (looked up) using **texture():**

```
gvec texture(gsampler sampler, vec texCoord[, float bias]);
```

- e.g. for a 2D texture with RGB values :

```
vec3 fragColor = texture(uSampler,
    vec2(textureCoordU, textureCoordV));
```

Image Processing: Loading an Image

See examples/hello-image

Exercise 5.1:
Red Channel

Send index.html to 3d@kbvis.com

KB-VIS
We Work In-Depth

DAY 7

Image Processing: Blurring

See examples/image-processing-2-blurring

Simple Horizontal Blur


```
<!-- fragment shader -->
<script id="2d-fragment-shader" type="x-shader/x-fragment">#version 300 es
precision mediump float;

// our texture
uniform sampler2D u_image;
uniform vec2 u_textureSize;

// the texCoords passed in from the vertex shader.
in vec2 v_texCoord;

out vec4 fragColor;

void main() {
   vec2 onePixel = vec2(1.0, 1.0) / u_textureSize;
    fragColor = (
        texture(u_image, v_texCoord) +
        texture(u_image, v_texCoord + vec2(onePixel.x, 0.0)) +
        texture(u_image, v_texCoord + vec2(-onePixel.x, 0.0))) / 3.0;
}
</script>
```

See examples/image-processing-2-blurring

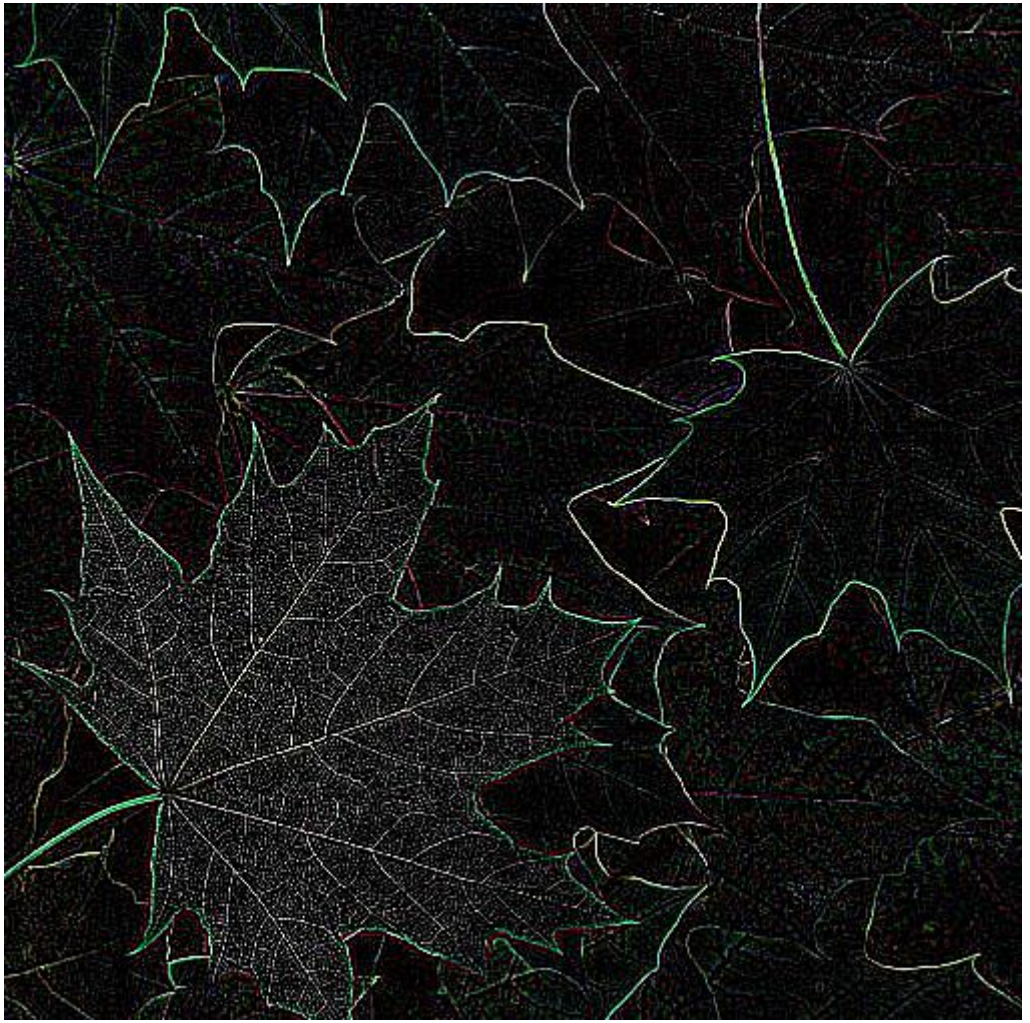
See image-processing-2-blurring/blurring-webgl2-magnified.html

Exercise:
Extend to 3x3 Blur

1	1	1
1	1	1
1	1	1

Send edited HTML to 3d@kbvis.com

Image Processing: Convolution



See examples/image-processing-3-convolution

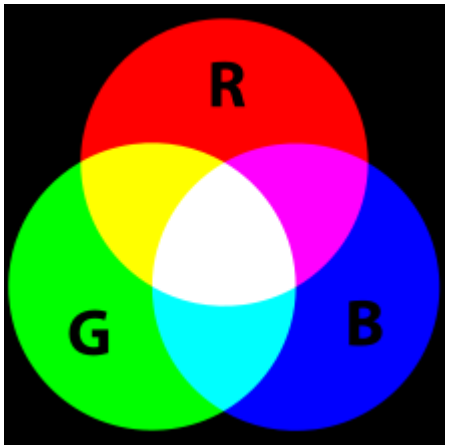
Convolution Kernels

Original	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge-Detect	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

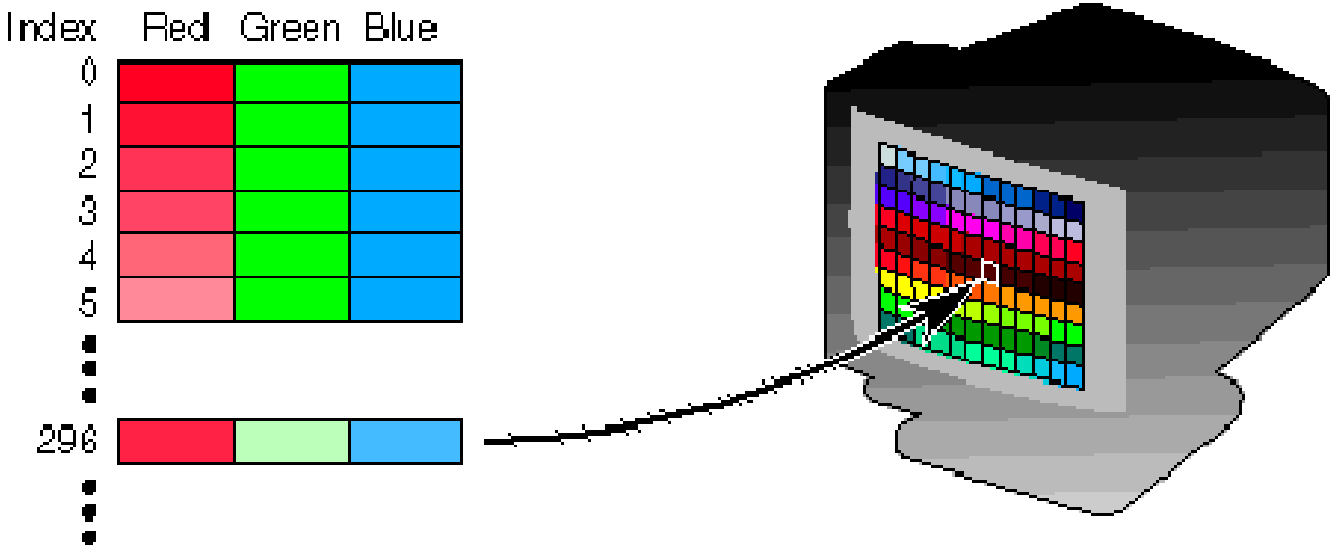
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Blur*	$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

Convolution kernels are essentially weighted averages of pixels

Color



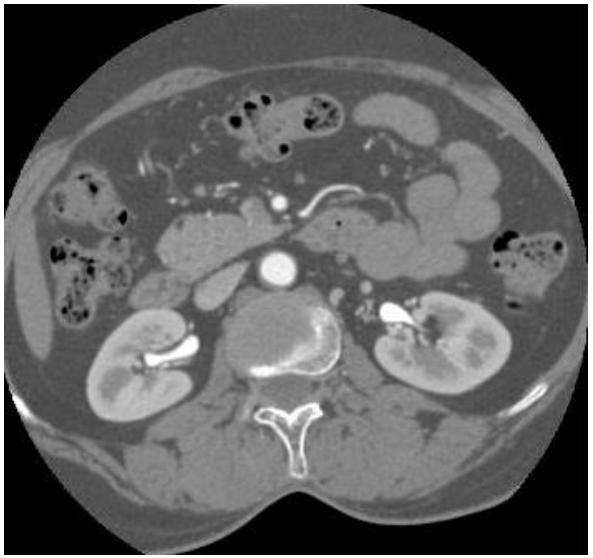
RGB – Additive Model



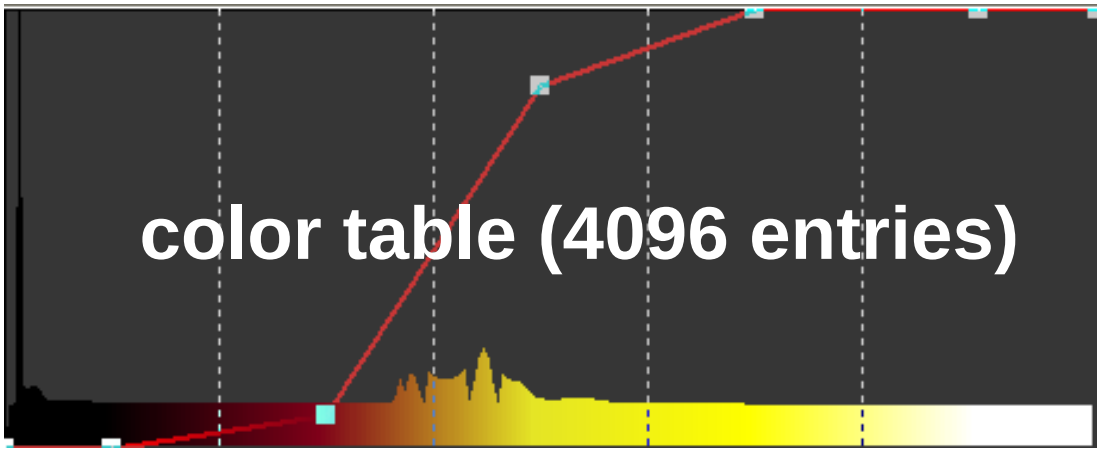
palette

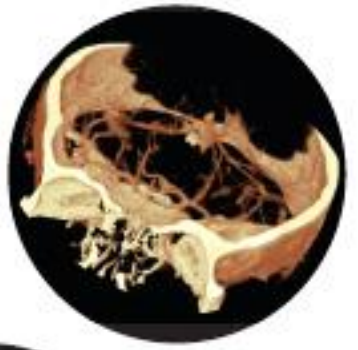
Color Index

Color Tables



Source image
(12 bit grayscale)

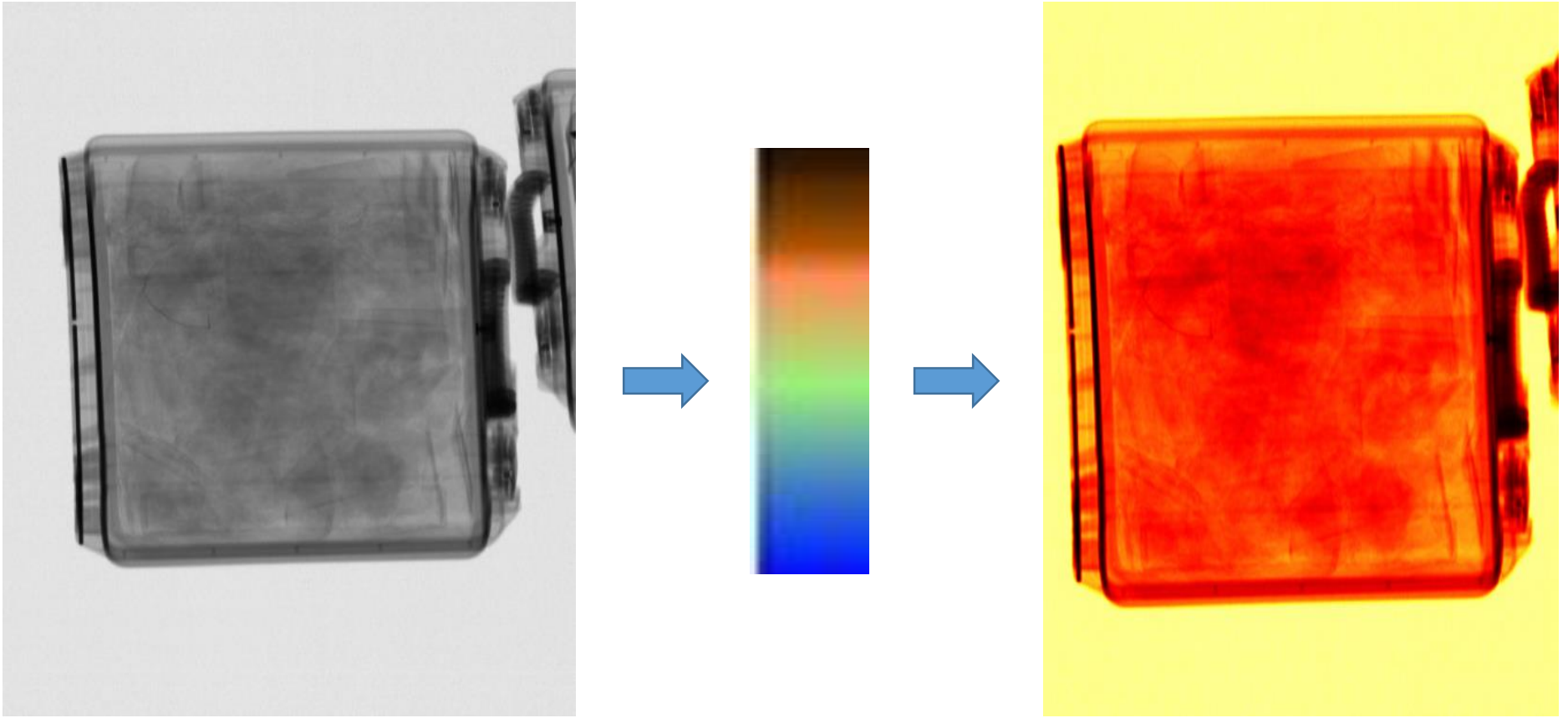




color index lookup:
`(unsigned int)` ➡ **`(R,G,B,A)`**

Example – Indexed Color with Lookup Table (LUT)

KB-VIS
We Work In-Depth



See examples/image-processing-4-color-index

```
var fragmentShaderSource = `#version 300 es
precision highp float;

// our texture
uniform sampler2D u_image;
uniform sampler2D u_colourTableTexture;

// the texCoords passed in from the vertex shader.
in vec2 v_texCoord;

out vec4 color;

void main() {
    // sample the source image (single-channel/grayscale/intensity/luminance)
    vec4 imageColour = texture(u_image, v_texCoord);

    // get index for looking up colour table
    int index = int(imageColour.r*255.0);
    ivec2 tableTexCoord = ivec2(index, 0);

    // sample (look-up) the colour table to get the final colour
    color = texelFetch(u_colourTableTexture, tableTexCoord, 0);
}
`;
```

texelFetch

Declaration

```
gvec4 texelFetch( gsampler2D sampler,
                  ivec2 P,
                  int lod);
```

Parameters

sampler

Specifies the sampler to which the texture from which texels will be retrieved is bound.

P

Specifies the texture coordinates at which texture will be sampled.

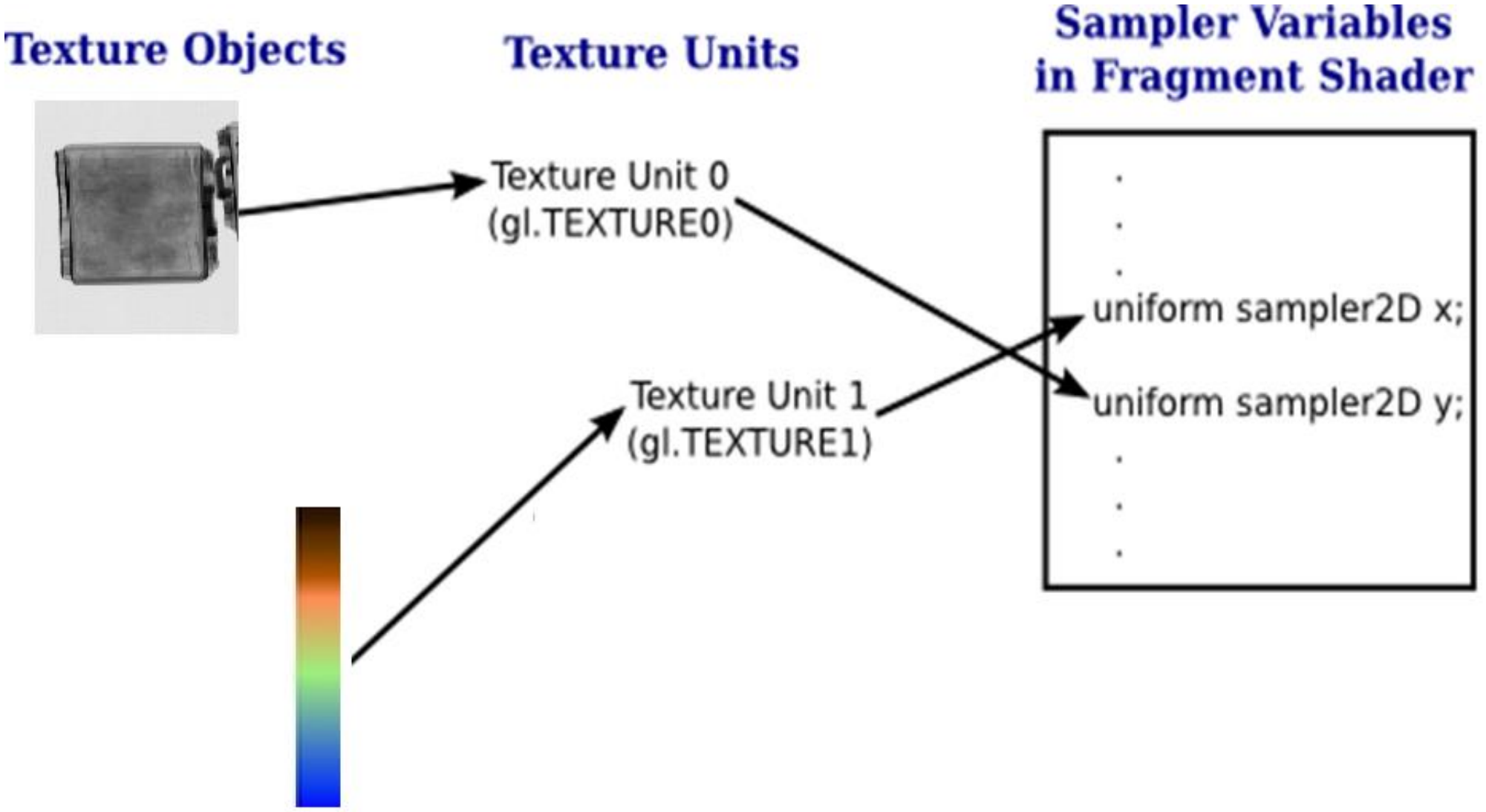
lod

If present, specifies the level-of-detail within the texture from which the texel will be fetched.

Description

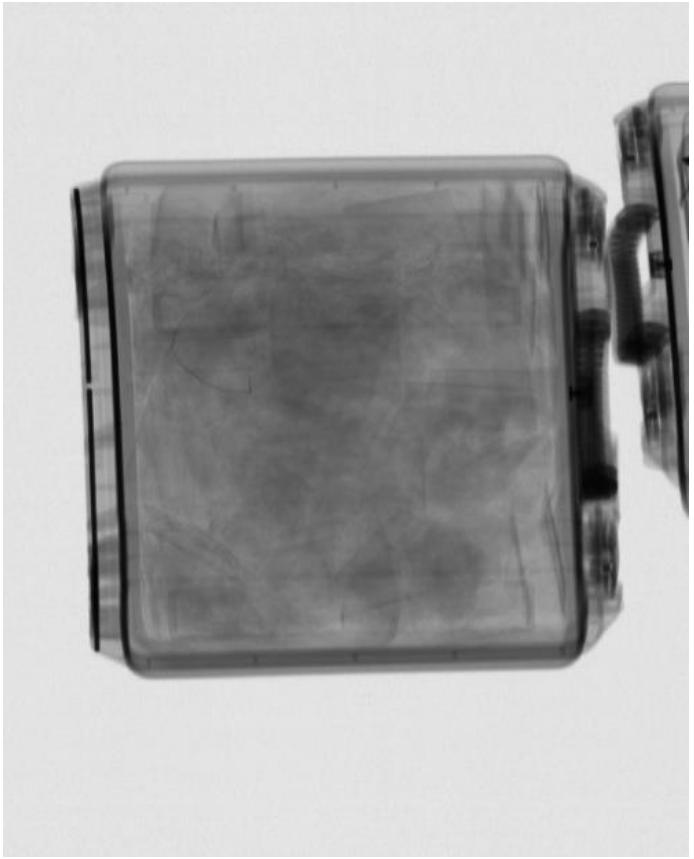
texelFetch performs a lookup of a single texel from texture coordinate *P* in the texture bound to *sampler*. The array layer is specified in the last component of *P* for array forms. The *lod* parameter (if present) specifies the level-of-detail from which the texel will be fetched.

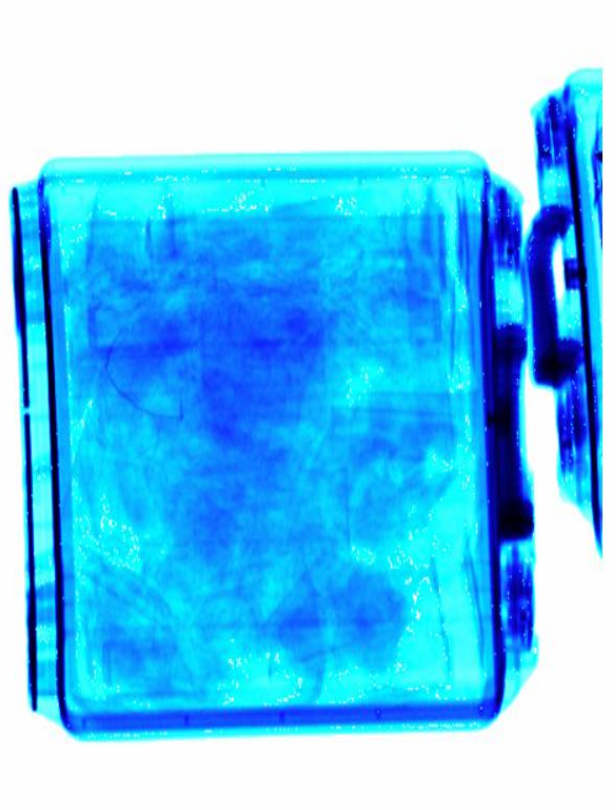
Texture Units Revisited



Exercise – Replace the LUT with a Blue - Cyan - White spectrum

KB-VIS
We Work In-Depth





Send your edited HTML to 3d@kbvis.com

DAY 8

Updated Content

- Updated slides:

http://kbvis.com/downloads/preso.zip

- Updated examples:

http://kbvis.com/downloads/kbvis-examples-webgl.zip

Exercise:
Extend to 3x3 Blur

1	1	1
1	1	1
1	1	1

Send edited HTML to 3d@kbvis.com

```
<script id="2d-fragment-shader" type="x-shader/x-fragment">#version 300 es
    precision mediump float;

    uniform sampler2D u_image;
    uniform vec2 u_textureSize;
    uniform float u_zoom;

    // the texCoords passed in from the vertex shader.
    in vec2 v_texCoord;

    out vec4 fragColor;

    void main() {
        vec2 onePixel = vec2(1.0, 1.0) / u_textureSize;
        // lookup left and right neighbours and average
        fragColor = (
            texture(u_image, v_texCoord) +
            texture(u_image, v_texCoord + vec2(-onePixel.x, -onePixel.y)) +
            texture(u_image, v_texCoord + vec2(-onePixel.x, onePixel.y)) +
            texture(u_image, v_texCoord + vec2(onePixel.x, -onePixel.y)) +
            texture(u_image, v_texCoord + vec2(onePixel.x, onePixel.y)) +
            texture(u_image, v_texCoord + vec2(0.0, onePixel.y)) +
            texture(u_image, v_texCoord + vec2(0.0, -onePixel.y)) +
            texture(u_image, v_texCoord + vec2(onePixel.x, 0.0)) +
            texture(u_image, v_texCoord + vec2(-onePixel.x, 0.0))
        ) / 9.0;
        // simple lookup
        //fragColor = texture(u_image, v_texCoord);
    }
</script>
```

Streaming Texture

See examples/image-processing-scrolling

```
void glTexSubImage2D( GLenum target,
                      GLint level,
                      GLint xoffset,
                      GLint yoffset,
                      GLsizei width,
                      GLsizei height,
                      GLenum format,
                      GLenum type,
                      const GLvoid * pixels);
```

Update the contents of the specified sub-image of the texture

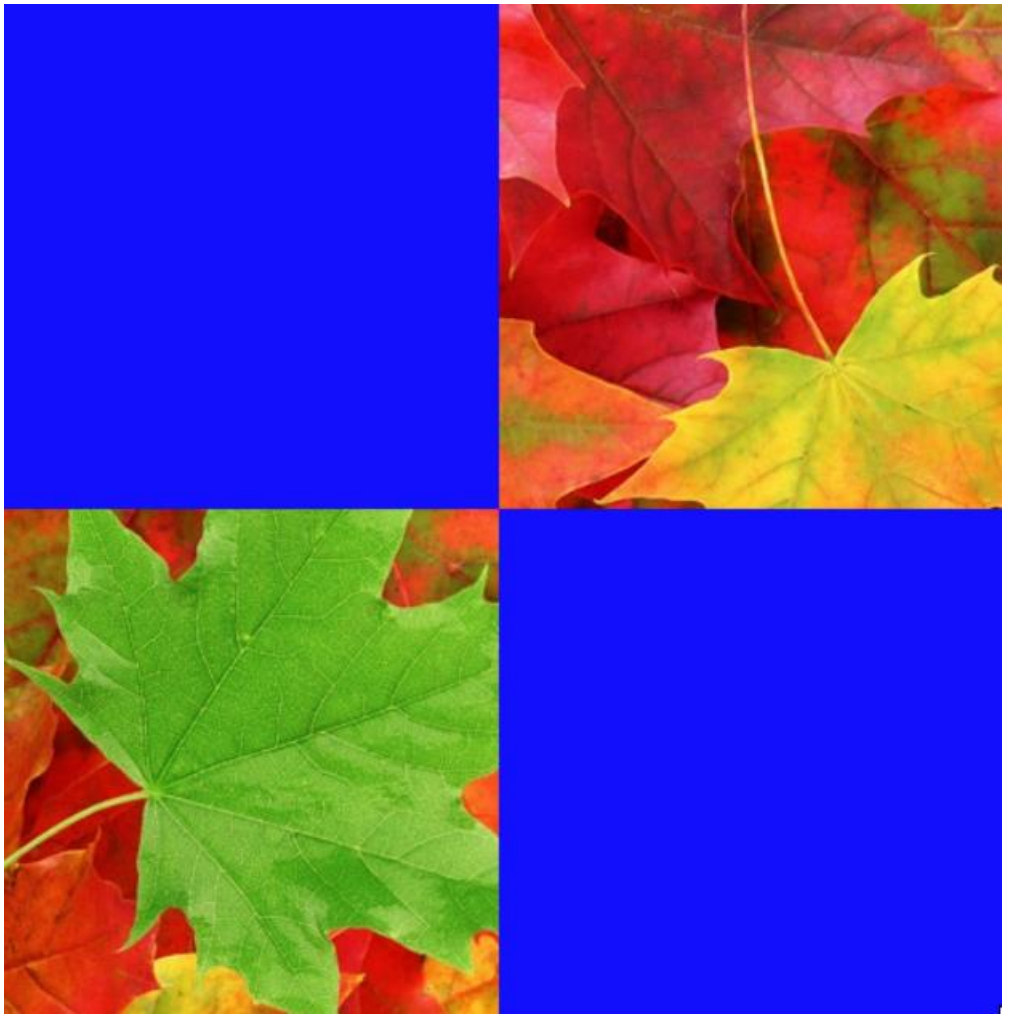
Updating a portion of the texture


```
function updateTexture()
{
    // we want to replace a bit of the texture from the right hand side
    var column = image.width - (frame * columnsToUpdate) % image.width - columnsToUpdate;

    // generate intensities for a grayscale ramp
    var intensity = (frame * columnsToUpdate * 0.05) % 256;
    for (var i = 0; i < image.height * columnsToUpdate; i++) {
        newColumn[i * 4 + 0] = intensity;
        newColumn[i * 4 + 1] = intensity;
        newColumn[i * 4 + 2] = intensity;
        newColumn[i * 4 + 3] = 255;
    }

    ++frame;

    var buffer = new Uint8Array(newColumn);

    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texSubImage2D(gl.TEXTURE_2D, 0, column, 0, columnsToUpdate, image.height,
        gl.RGBA, gl.UNSIGNED_BYTE, buffer);
}
```

Scroll in the Shader


```
<!-- fragment shader -->
<script id="2d-fragment-shader" type="x-shader/x-fragment">#version 300 es // has to be on first line
    precision mediump float;

    // our texture
    uniform sampler2D u_image;
    uniform sampler2D u_image2;
    uniform vec2 u_textureSize;
    uniform float u_frame;
    uniform float u_columnCount;

    // the texCoords passed in from the vertex shader.
    in vec2 v_texCoord;

    out vec4 fragColor;

    void main() {
        float pixelStep = u_columnCount / u_textureSize.x;
        fragColor = texture(u_image, vec2(v_texCoord.x - u_frame*pixelStep, v_texCoord.y));
    }
</script>
```

Scissor Test



examples/image-processing-scissor-test

glScissor


```
void glScissor( GLint x,
                GLint y,
                GLsizei width,
                GLsizei height);
```

Parameters

x, *y*

Specify the lower left corner of the scissor box. Initially (0, 0).

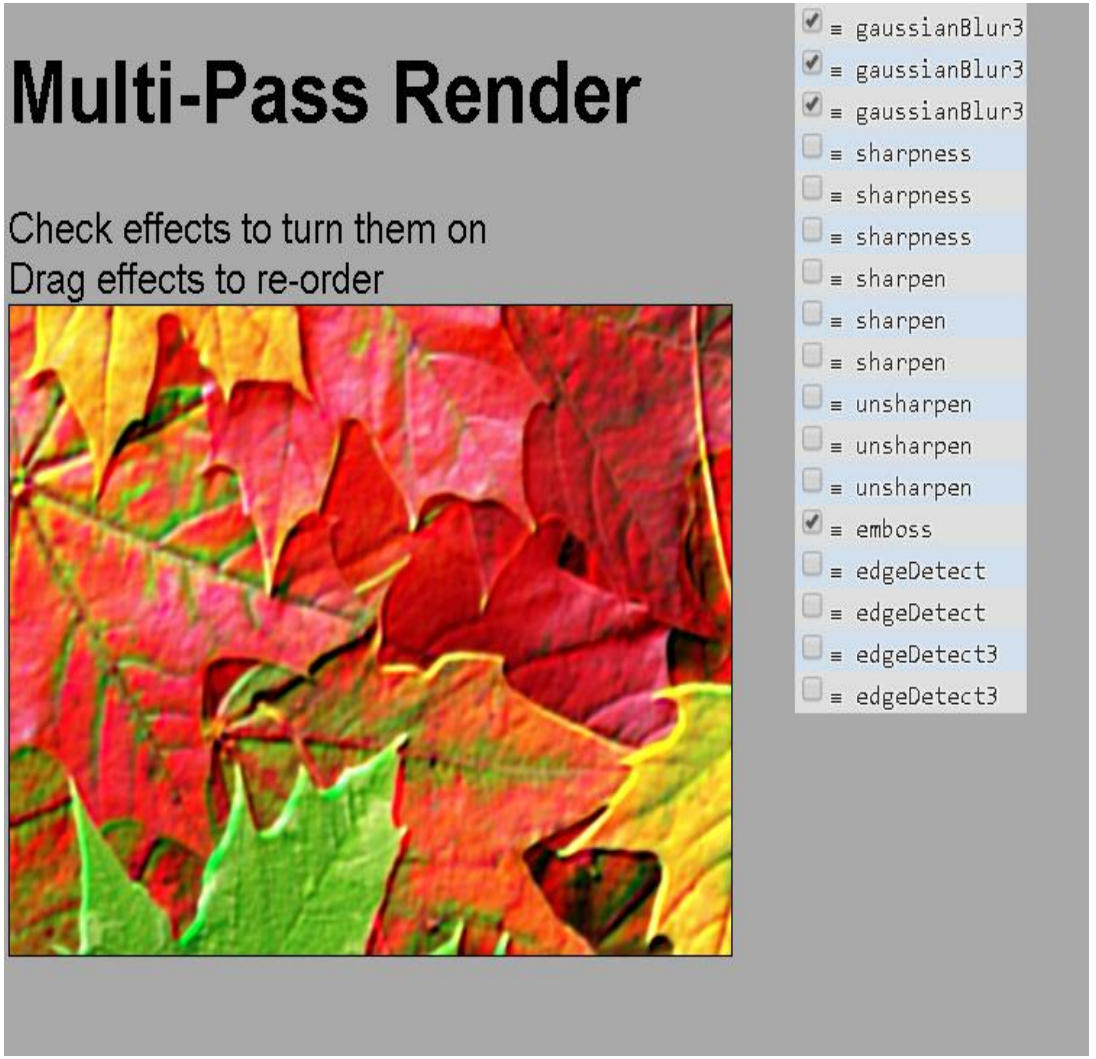
width, *height*

Specify the width and height of the scissor box. When a GL context is first attached to a window, *width* and *height* are set to the dimensions of that window.

To enable and disable the scissor test, call glEnable and glDisable with argument GL_SCISSOR_TEST.

```
gl.clearColor(0, 0, 1, 1);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

// set the scissor rectangle
gl.scissor(0, 0, canvas.width / 2, canvas.height / 2);
gl.enable(gl.SCISSOR_TEST);
gl.clearColor(0, 0, 0, 1);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
// draw
gl.drawArrays(gl.TRIANGLES, 0, 6);

// set the scissor rectangle
gl.scissor(canvas.width / 2, canvas.height / 2, canvas.width / 2, canvas.height / 2);
// draw
gl.drawArrays(gl.TRIANGLES, 0, 6);

gl.disable(gl.SCISSOR_TEST);
```

Off-Screen Rendering



examples/image-processing-multiple-shaders

FBOs

- **Framebuffer Objects –** used to render to off-screen targets
- Can be used for post-processing effects
- Render to texture
- Attach images of type color, depth, stencil
- **glGenFramebuffers / glCreateFramebuffer**
- **glBindFramebuffer**

Name

glCreateFramebuffers — create framebuffer objects

C Specification

```
void glCreateFramebuffers( GLsizei n,
                           GLuint *ids);
```

Parameters

n

Number of framebuffer objects to create.

framebuffers

Specifies an array in which names of the new framebuffer objects are stored.

Render To Texture

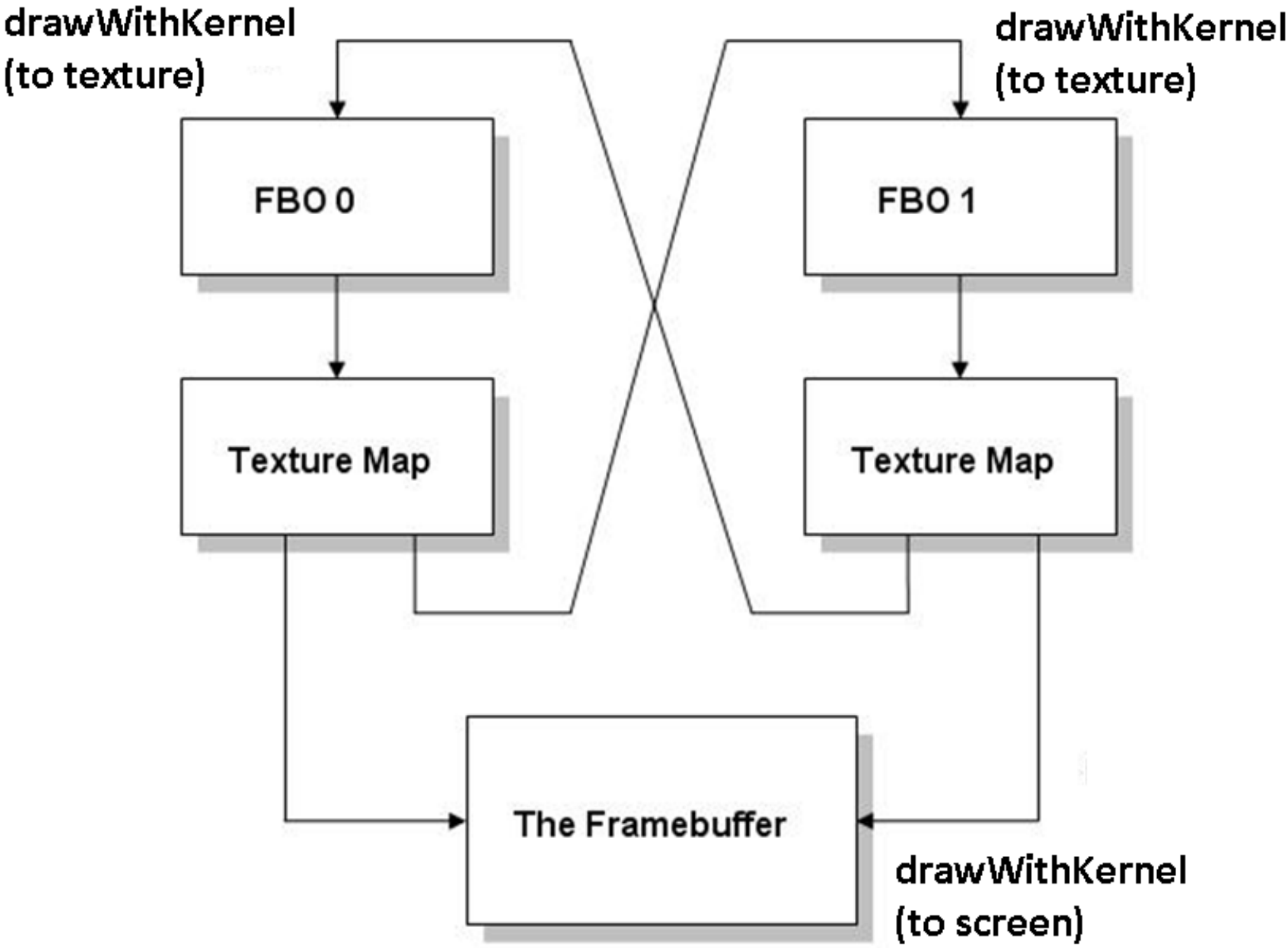
Name

glFramebufferTexture2D — attach a texture image to a framebuffer object

C Specification

```
void glFramebufferTexture2D( GLenum target,
                             GLenum attachment,
                             GLenum textarget,
                             GLuint texture,
                             GLint level);
```

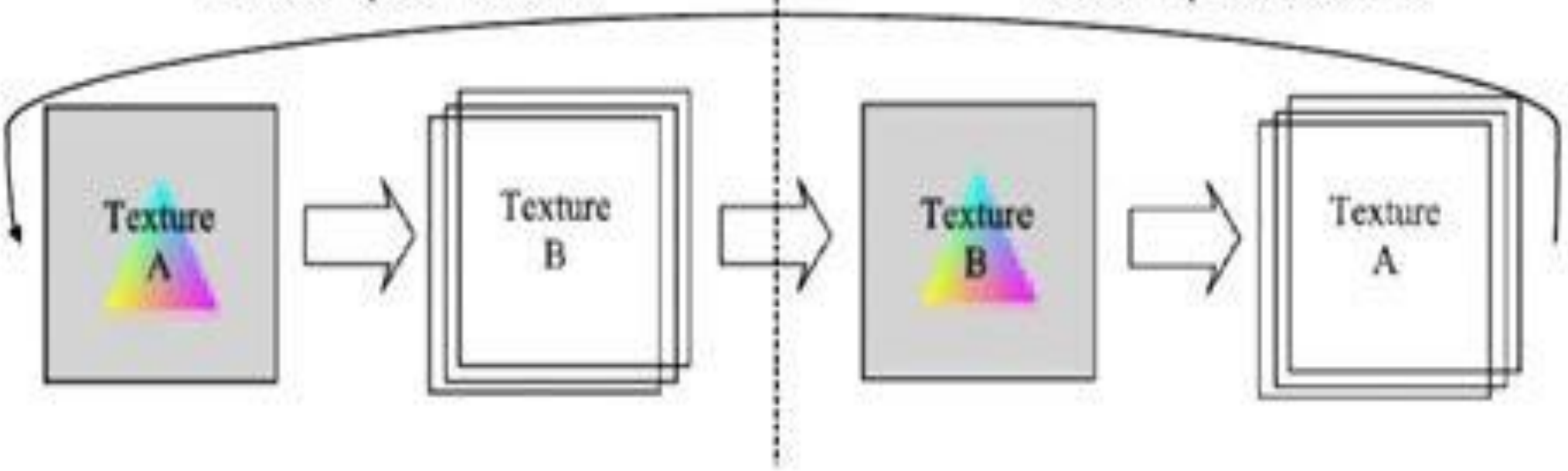
```
gl.framebufferTexture2D(
        gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, texture, 0);
```


drawWithKernel
(to texture)
drawWithKernel
(to texture)
FBO 0
FBO 1
Texture Map
Texture Map
The Framebuffer
drawWithKernel
(to screen)

Multi Pass Processing

1 pass:
attach fbo with B
use shader
render quad with A

2 pass:
attach fbo with A
use shader
render quad with B

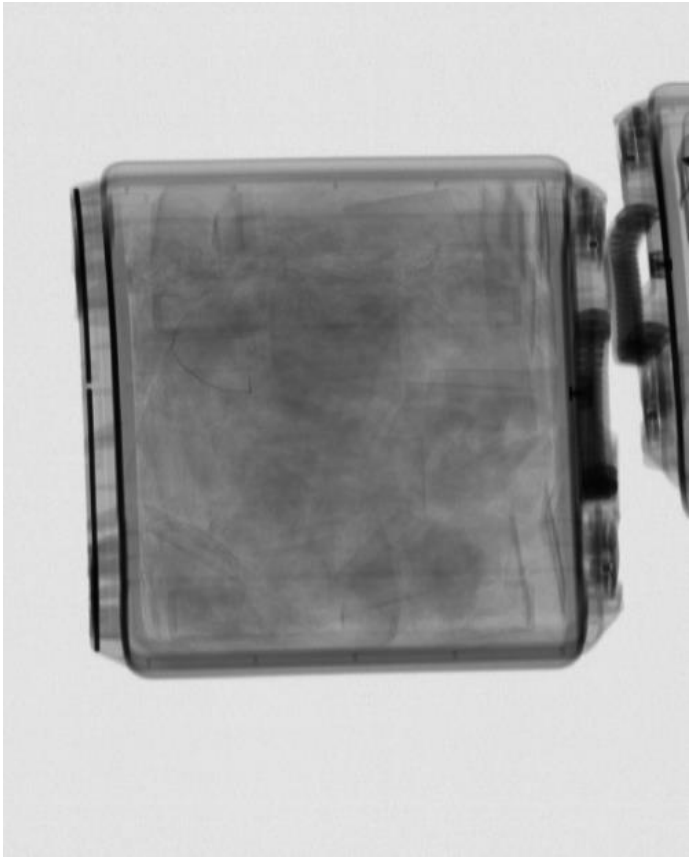


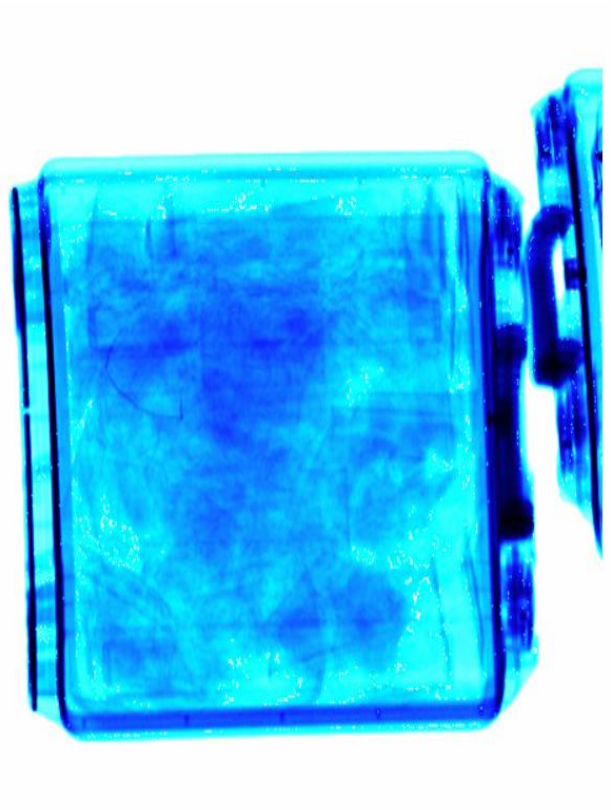
A post-process quad with texture A is rendered into texture B via an FBO. Texture B is then rendered via another post-process quad into texture A, ready for another repetition of the process.

DAY 9

Exercise – Replace the LUT with a Blue - Cyan - White spectrum

KB-VIS
We Work In-Depth

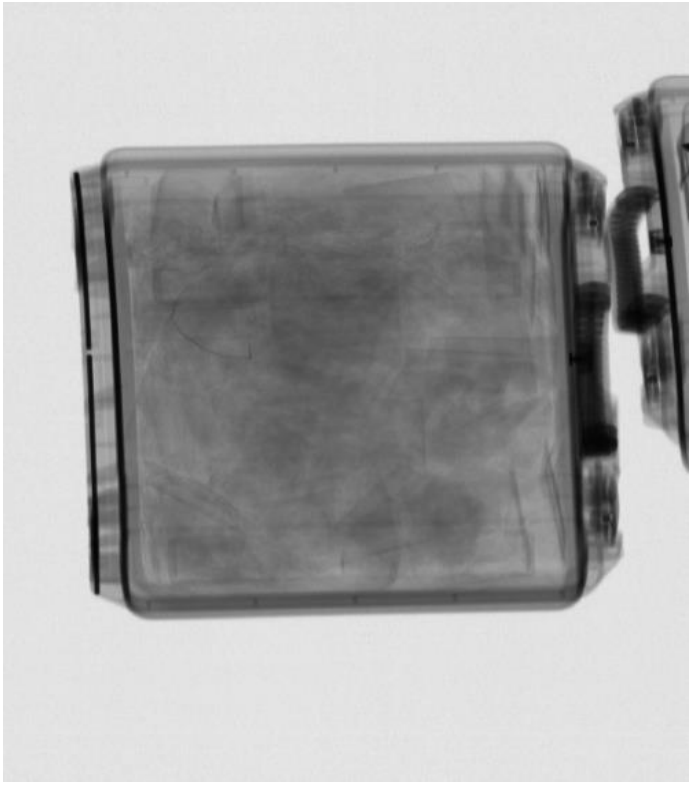


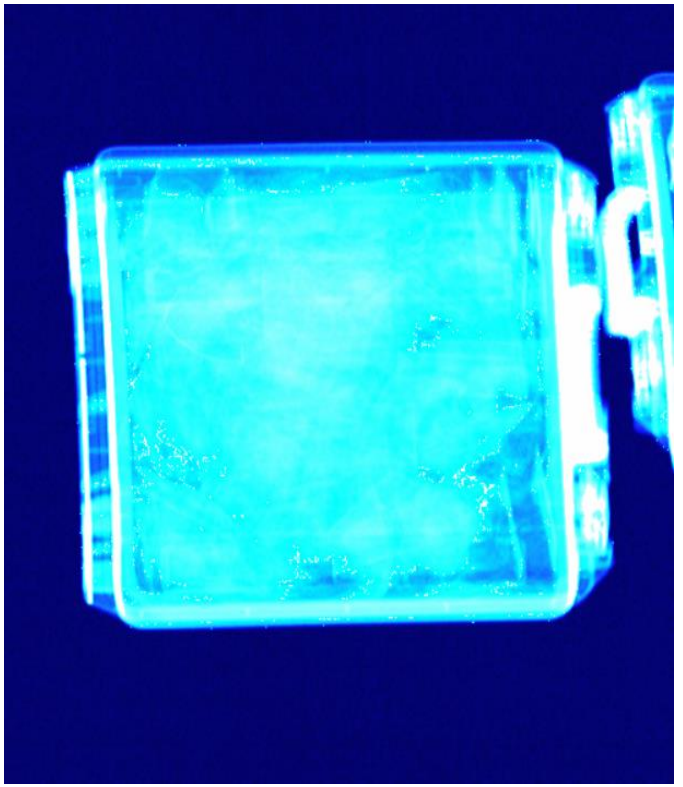


Send your edited HTML to 3d@kbvis.com

Follow-up Exercise – Invert the LUT lookup to White => Cyan => Blue spectrum

KB-VIS
We Work In-Depth





Modify only this line in the shader:

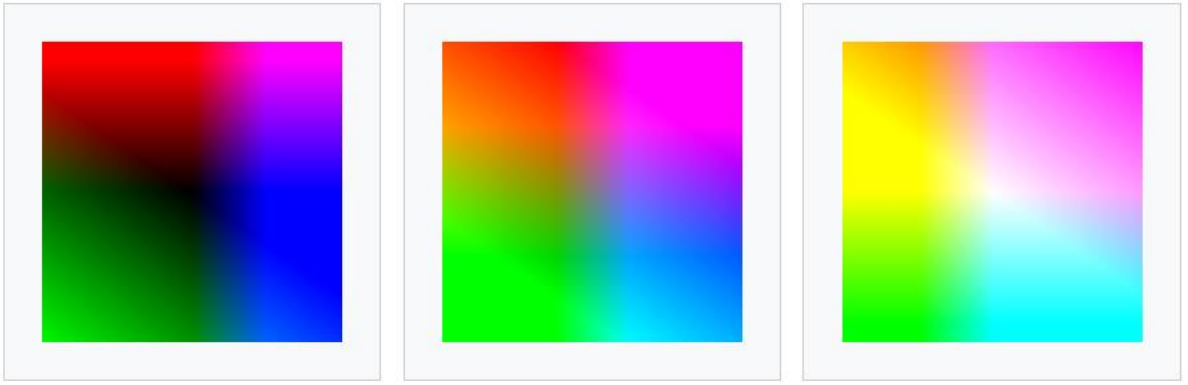
```
// get index for looking up colour table
int index = int((imageColour.r)*255.0) + u_lutOffset;
```

Send your edited HTML to 3d@kbvis.com

Exercise: Streaming Texture – Reverse Scroll Direction (Right to Left)

Send edited HTML to 3d@kbvis.com

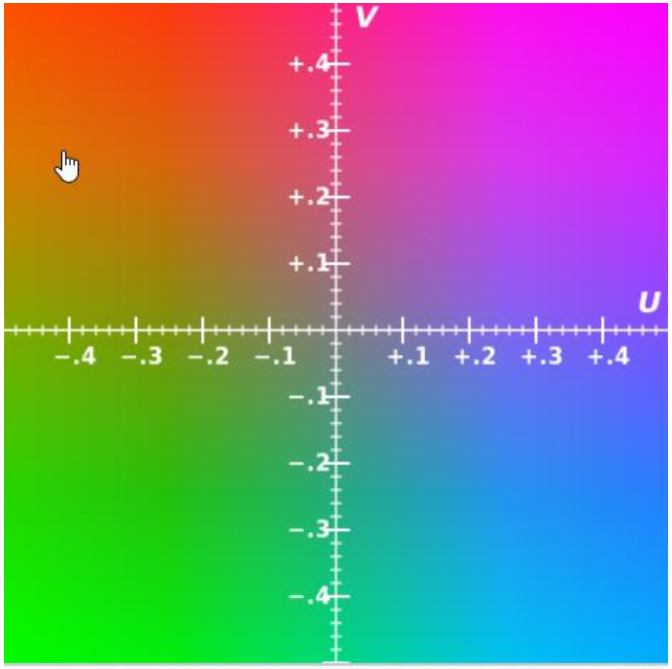
Image Processing: YUV Color Space



Y′ value of 0 | Y′ value of 0.5 | Y′ value of 1

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.2126 & 0.7152 & 0.0722 \\ -0.09991 & -0.33609 & 0.436 \\ 0.615 & -0.55861 & -0.05639 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.28033 \\ 1 & -0.21482 & -0.38059 \\ 1 & 2.12798 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}$$



Example of U-V color plane, Y′ value = 0.5, represented within RGB color gamut

See examples/example-yuv

YUV Colour Planes

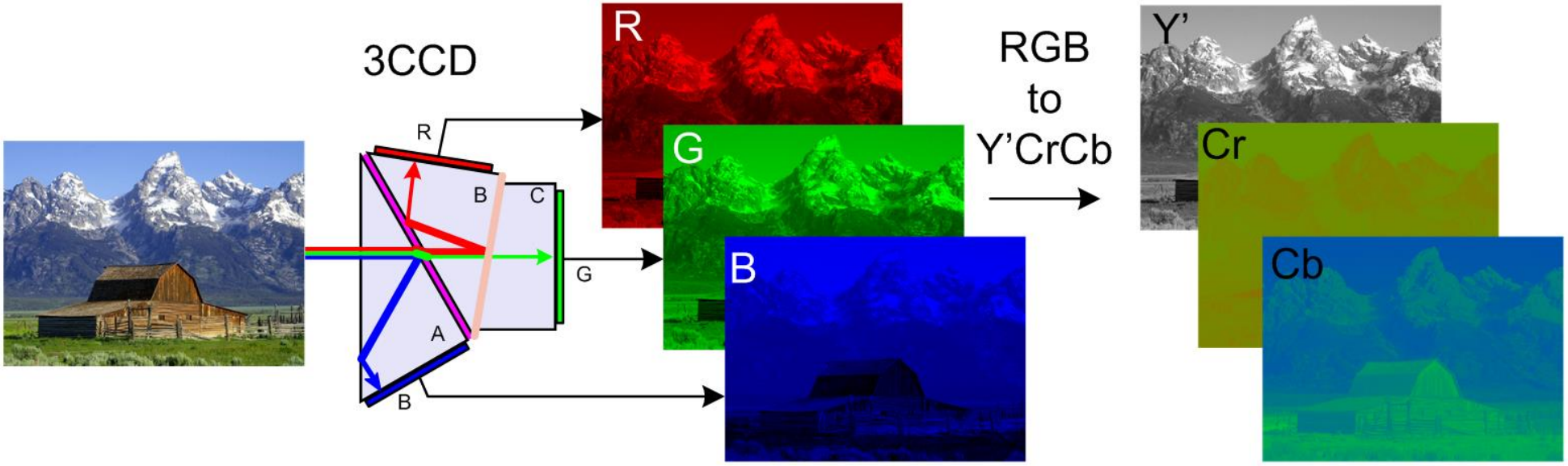


Image Processing: YUV Example

- Y, U, V planes are uploaded as luminance textures (single channel)
- The Y, U, V components are then combined into a RGB value by the fragment shader

```
// Y, Cb, and Cr planes are uploaded as LUMINANCE textures.
float fY = texture2D(uTextureY, vLumaPosition).x;
float fCb = texture2D(uTextureCb, vChromaPosition).x;
float fCr = texture2D(uTextureCr, vChromaPosition).x;
gl_FragColor = vec4(
  fYmul + 1.59602734375 * fCr - 0.87078515625,
  fYmul - 0.39176171875 * fCb - 0.81296875 * fCr + 0.52959375,
  fYmul + 2.017234375   * fCb - 1.081390625,
  1
);
```

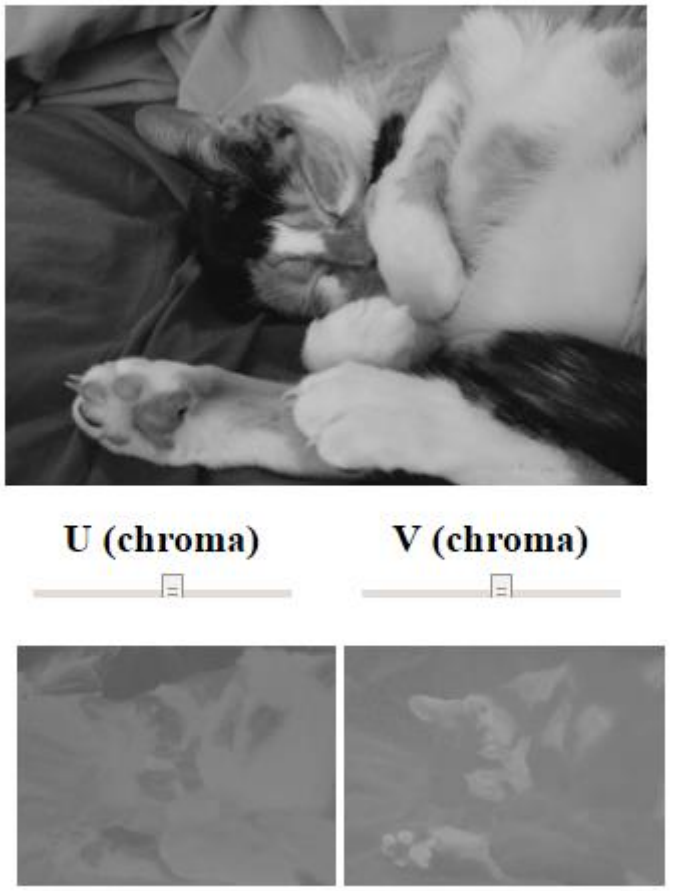
See example/example-yuv

Luminance Textures

- Single channel
- Used for grayscale or with colour LUT
- Not needed now – can replace with GL_RED

```
gl.texImage2D(
    gl.TEXTURE_2D,
    0, // mip level
    gl.LUMINANCE, // internal format
    texWidth,
    height,
    0, // border
    format, // format
    gl.UNSIGNED_BYTE, //type
    data // data!
);
```

Image Processing: YUV Demo



See examples/example-yuv

or

https://brion.github.io/yuv-canvas/demo.html

OpenGL Lighting with Shaders

Shaders allow flexible, advanced models

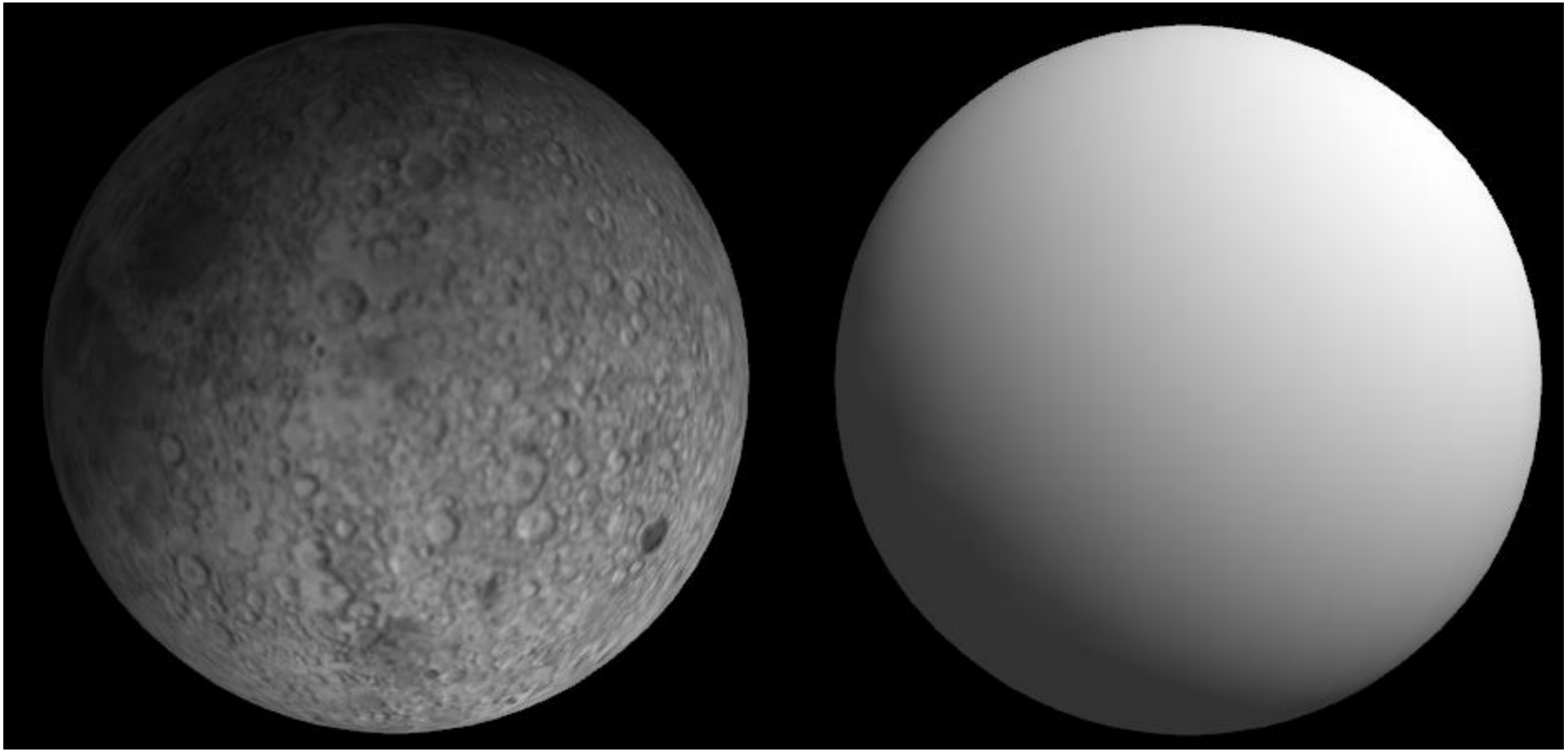
- GLSL built-in variables(compatibility profile, not in WebGL):

```
uniform gl_LightSourceParameters gl_LightSource[gl_MaxLights];
uniform gl_MaterialParameters gl_FrontMaterial;
uniform gl_MaterialParameters gl_BackMaterial;
```

- GLSL Core - write your own lighting model
  - Directional Lights – per vertex / per-pixel
  - Point Lights
  - Spotlights
  - Anything else you can think of !

Example - Lighting

(Directional Lighting, Per-Vertex Diffuse Shading)



See examples/directional-lighting

The Vertex Shader


```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexNormal;
    attribute vec2 aTextureCoord;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    uniform mat3 uNMatrix;

    uniform vec3 uAmbientColor;
    uniform vec3 uLightingDirection;
    uniform vec3 uDirectionalColor;

    varying vec2 vTextureCoord;
    varying vec3 vLightWeighting;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vTextureCoord = aTextureCoord;

        vec3 transformedNormal = uNMatrix * aVertexNormal;
        float directionalLightWeighting = max(dot(transformedNormal, uLightingDirection), 0.0);
        vLightWeighting = uAmbientColor + uDirectionalColor * directionalLightWeighting;
    }
</script>
```

- Compute the illumination at the vertex by applying the directional diffuse lighting model
- Multiply the diffuse light colour by the illumination value to compute the diffuse shade
- Add the ambient and diffuse values to compute the final lighting at the vertex
- The lighting value at vLightWeighting (varying) is then interpolated across fragments

The Fragment Shader


```
<script id="shader-fs" type="x-shader/x-fragment">
    precision mediump float;

    varying vec2 vTextureCoord;
    varying vec3 vLightWeighting;

    uniform sampler2D uSampler;

    void main(void) {
        vec4 textureColor = texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
        gl_FragColor = vec4(textureColor.rgb * vLightWeighting, textureColor.a);
    }
</script>
```

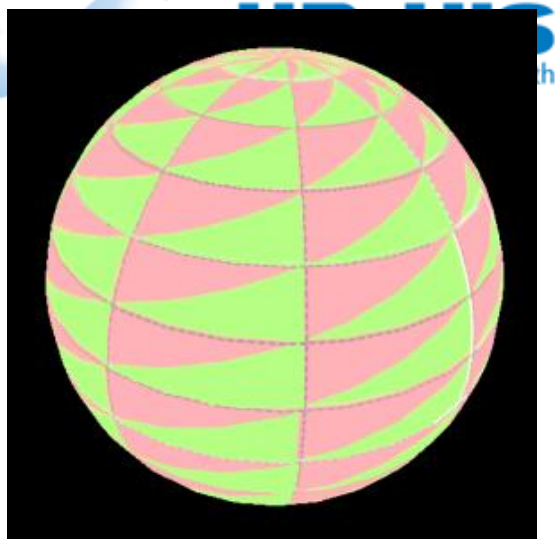
- The shaded colour from the vertex shader is interpolated and passed into the fragment shader through vLightWeighting
- The shaded colour is modulated by the texture map colour and output as the final fragment colour

Lighting Parameters

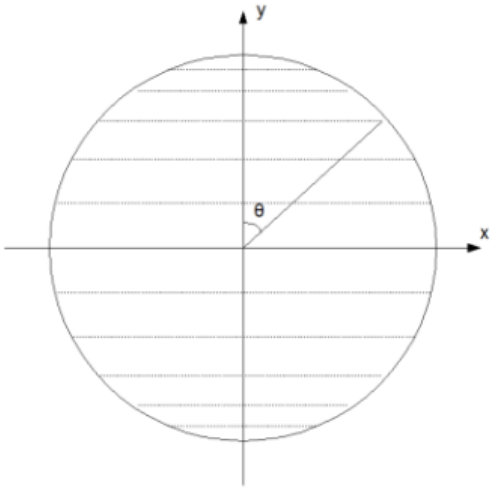

```
function initShaders() {
    var fragmentShader = getShader(gl, "shader-fs");
    var vertexShader = getShader(gl, "shader-vs");

    shaderProgram = gl.createProgram();
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram(shaderProgram);

    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
        alert("Could not initialise shaders");
    }

    gl.useProgram(shaderProgram);

    shaderProgram.vertexPositionAttribute = gl.getAttribLocation(shaderProgram, "aVertexPosition");
    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

    shaderProgram.textureCoordAttribute = gl.getAttribLocation(shaderProgram, "aTextureCoord");
    gl.enableVertexAttribArray(shaderProgram.textureCoordAttribute);

    shaderProgram.vertexNormalAttribute = gl.getAttribLocation(shaderProgram, "aVertexNormal");
    gl.enableVertexAttribArray(shaderProgram.vertexNormalAttribute);

    shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram, "uPMatrix");
    shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram, "uMVMatrix");
    shaderProgram.nMatrixUniform = gl.getUniformLocation(shaderProgram, "uNMatrix");
    shaderProgram.samplerUniform = gl.getUniformLocation(shaderProgram, "uSampler");
    shaderProgram.useLightingUniform = gl.getUniformLocation(shaderProgram, "uUseLighting");
    shaderProgram.ambientColorUniform = gl.getUniformLocation(shaderProgram, "uAmbientColor");
    shaderProgram.lightingDirectionUniform = gl.getUniformLocation(shaderProgram, "uLightingDirection");
    shaderProgram.directionalColorUniform = gl.getUniformLocation(shaderProgram, "uDirectionalColor");
}
```

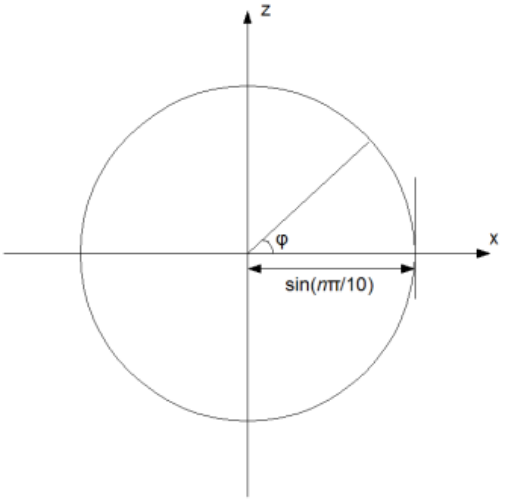
gl.**getAttribLocation**() - returns the “address” of the specified shader variable
gl.**enableVertexAttribArray**() - the values in the vertex attribute array will be accessed and used for rendering when calls are made to glDrawArrays or glDrawElements
gl.**getUniformLocation**() - returns the “address” of the specified shader uniform, which can then be used to set the value.

Sphere Geometry



```
function initBuffers() {
        var latitudeBands = 30;
        var longitudeBands = 30;
        var radius = 2;

        var vertexPositionData = [];
        var normalData = [];
        var textureCoordData = [];
        for (var latNumber=0; latNumber <= latitudeBands; latNumber++) {
            var theta = latNumber * Math.PI / latitudeBands;
            var sinTheta = Math.sin(theta);
            var cosTheta = Math.cos(theta);

            for (var longNumber=0; longNumber <= longitudeBands; longNumber++) {
                var phi = longNumber * 2 * Math.PI / longitudeBands;
                var sinPhi = Math.sin(phi);
                var cosPhi = Math.cos(phi);

                var x = cosPhi * sinTheta;
                var y = cosTheta;
                var z = sinPhi * sinTheta;
                var u = 1 - (longNumber / longitudeBands);
                var v = 1 - (latNumber / latitudeBands);

                normalData.push(x);
                normalData.push(y);
                normalData.push(z);
                textureCoordData.push(u);
                textureCoordData.push(v);
                vertexPositionData.push(radius * x);
                vertexPositionData.push(radius * y);
                vertexPositionData.push(radius * z);
            }
        }

        var indexData = [];
        for (var latNumber=0; latNumber < latitudeBands; latNumber++) {
            for (var longNumber=0; longNumber < longitudeBands; longNumber++) {
                var first = (latNumber * (longitudeBands + 1)) + longNumber;
                var second = first + longitudeBands + 1;
                indexData.push(first);
                indexData.push(second);
                indexData.push(first + 1);

                indexData.push(second);
                indexData.push(second + 1);
                indexData.push(first + 1);
            }
        }
}
```





Setting Shader Uniforms

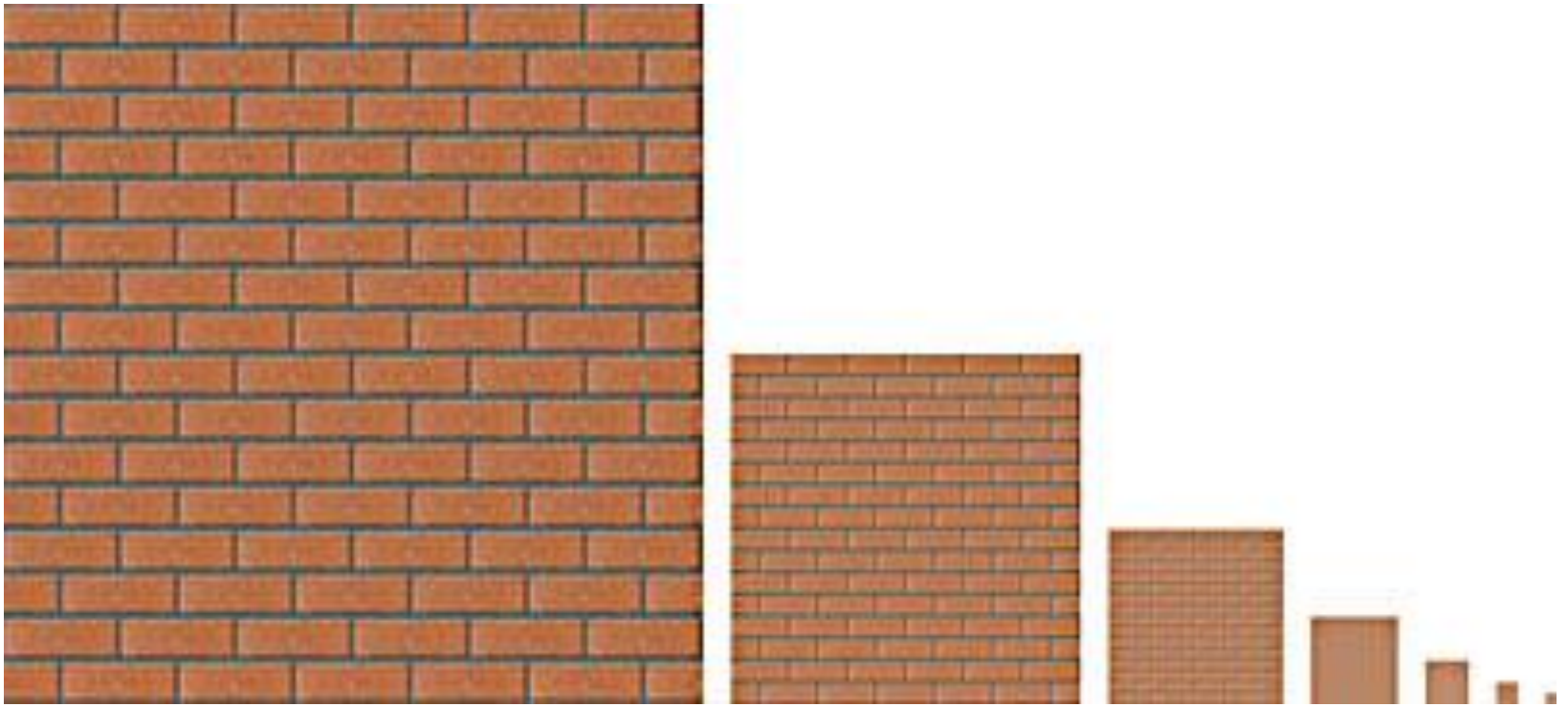

```
function drawScene() {
        ...
        var lighting = document.getElementById("lighting").checked;
        gl.uniform1i(shaderProgram.useLightingUniform, lighting);
    gl.uniform3f(
        shaderProgram.ambientColorUniform,
        parseFloat(document.getElementById("ambientR").value),
        parseFloat(document.getElementById("ambientG").value),
        parseFloat(document.getElementById("ambientB").value)
      );

        var lightingDirection = [
        parseFloat(document.getElementById("lightDirectionX").value),
        parseFloat(document.getElementById("lightDirectionY").value),
        parseFloat(document.getElementById("lightDirectionZ").value)
      ];
        var adjustedLD = vec3.create();
        vec3.normalize(lightingDirection, adjustedLD);
        vec3.scale(adjustedLD, -1);
    gl.uniform3fv(shaderProgram.lightingDirectionUniform, adjustedLD);

        gl.uniform3f(
        shaderProgram.directionalColorUniform,
        parseFloat(document.getElementById("directionalR").value),
        parseFloat(document.getElementById("directionalG").value),
        parseFloat(document.getElementById("directionalB").value)
        );
```

glUniform — specify the value of a uniform variable for the current shader program object
gl.uniform3f() - takes 3 float arguments
gl.uniform3fv() - takes an array of 3 floats
Many other variations – see OpenGL documentation

Draw Everything


```
    ...

    mat4.identity(mvMatrix);
    mat4.translate(mvMatrix, [0, 0, -6]);
    mat4.multiply(mvMatrix, moonRotationMatrix);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, moonTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
            moonVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
            moonVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, moonVertexNormalBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute,
            moonVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, moonVertexIndexBuffer);
    setMatrixUniforms();
    gl.drawElements(gl.TRIANGLES, moonVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
}
```

Mipmapping



- Use low-res version of texture when fragment coverage is low
- Mip map resolution is the “level” in GL texture API (0 is full-res)

Per-Fragment Lighting

(Specular Lighting, Per-Fragment Shading)

See examples/perfragment-lighting-json

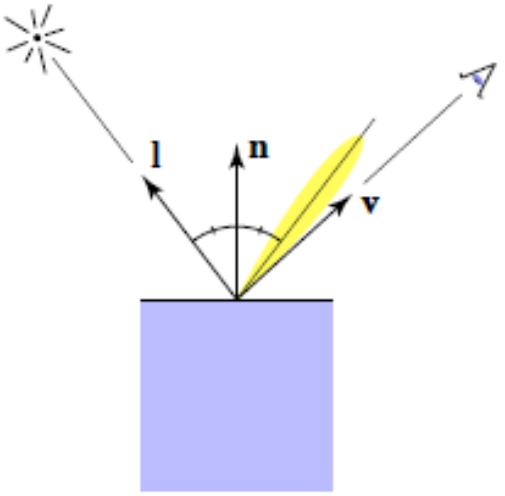
The Vertex Shader


```
<script id="per-fragment-lighting-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec3 aVertexNormal;
    attribute vec2 aTextureCoord;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;
    uniform mat3 uNMatrix;

    varying vec2 vTextureCoord;
    varying vec3 vTransformedNormal;
    varying vec4 vPosition;


    void main(void) {
        vPosition = uMVMatrix * vec4(aVertexPosition, 1.0);
        gl_Position = uPMatrix * vPosition;
        vTextureCoord = aTextureCoord;
        vTransformedNormal = uNMatrix * aVertexNormal;
    }
</script>
```

- As usual, output the transformed vertex position and the vertex's texture coordinate
- output the vertex's normal vector for use in the fragment shader

The Fragment Shader


```
<script id="per-fragment-lighting-fs" type="x-shader/x-fragment">
    varying vec2 vTextureCoord;
    varying vec3 vTransformedNormal;
    varying vec4 vPosition;

    uniform float uMaterialShininess;
    uniform vec3 uAmbientColor;
    uniform vec3 uPointLightingLocation;
    uniform vec3 uPointLightingSpecularColor;
    uniform vec3 uPointLightingDiffuseColor;
    uniform sampler2D uSampler;

    void main(void) {
        vec3 lightDirection = normalize(uPointLightingLocation - vPosition.xy
        vec3 normal = normalize(vTransformedNormal);
        vec3 eyeDirection = normalize(-vPosition.xyz);
        vec3 reflectionDirection = reflect(-lightDirection, normal);

        Float specularLightWeighting = pow(max(dot(reflectionDirection, eyeDirection), 0.0),

        uMaterialShininess);

        float diffuseLightWeighting = max(dot(normal, lightDirection), 0.0);
        vec3 lightWeighting = uAmbientColor
                + uPointLightingSpecularColor * specularLightWeighting
                + uPointLightingDiffuseColor * diffuseLightWeighting;

        vec4 fragmentColor = texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
        gl_FragColor = vec4(fragmentColor.rgb * lightWeighting, fragmentColor.a);
    }
</script>
```



Loading a JSON Model


```
function handleLoadedTeapot(teapotData) {
    teapotVertexNormalBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, teapotVertexNormalBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(teapotData.vertexNormals), gl.STATIC_DRAW);
    teapotVertexNormalBuffer.itemSize = 3;
    teapotVertexNormalBuffer.numItems = teapotData.vertexNormals.length / 3;

    teapotVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, teapotVertexTextureCoordBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(teapotData.vertexTextureCoords), gl.STATIC_DRAW);
    teapotVertexTextureCoordBuffer.itemSize = 2;
    teapotVertexTextureCoordBuffer.numItems = teapotData.vertexTextureCoords.length / 2;

    teapotVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, teapotVertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(teapotData.vertexPositions), gl.STATIC_DRAW);
    teapotVertexPositionBuffer.itemSize = 3;
    teapotVertexPositionBuffer.numItems = teapotData.vertexPositions.length / 3;

    teapotVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, teapotVertexIndexBuffer);
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(teapotData.indices), gl.STATIC_DRAW);
    teapotVertexIndexBuffer.itemSize = 1;
    teapotVertexIndexBuffer.numItems = teapotData.indices.length;
}

function loadTeapot() {
    var request = new XMLHttpRequest();
    request.open("GET", "Teapot.json");
    request.onreadystatechange = function () {
        if (request.readyState == 4) {
            handleLoadedTeapot(JSON.parse(request.responseText));
        }
    }
    request.send();
}
```

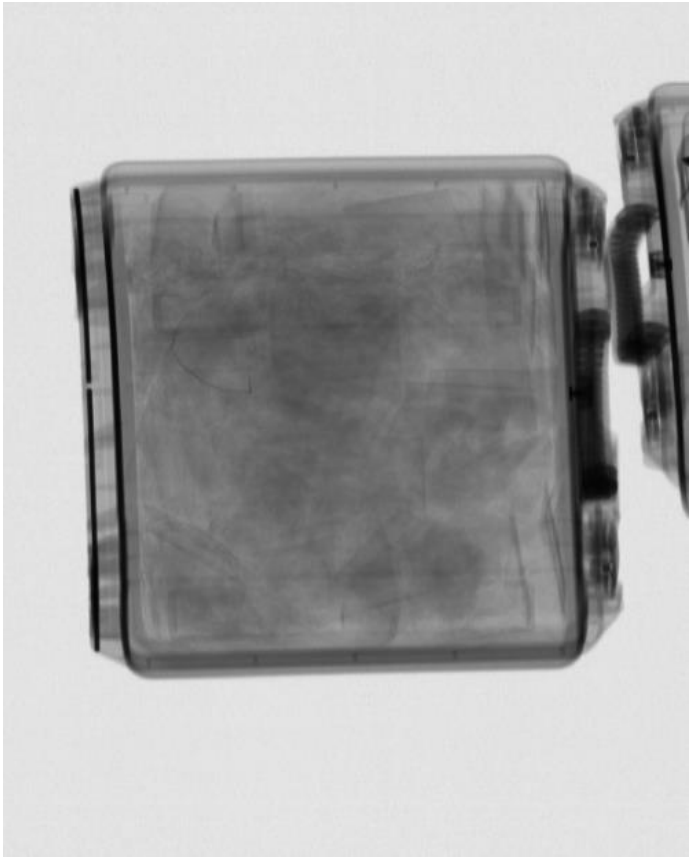
- JSON (JavaScript Object Notation) - lightweight data-interchange format
- Various utilities available to load/export/convert JSON and other data formats

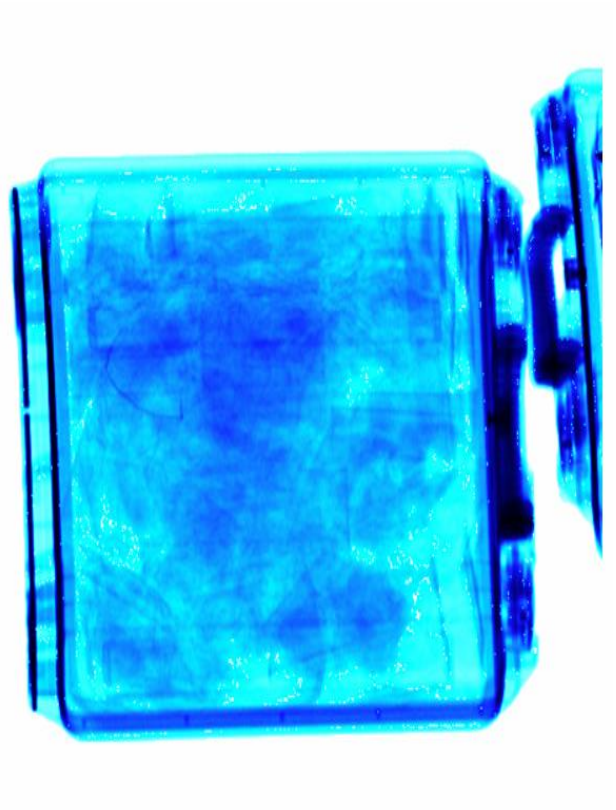
KB-VIS
We Work In-Depth

DAY 10

Exercise – Replace the LUT with a Blue - Cyan - White spectrum

KB-VIS
We Work In-Depth



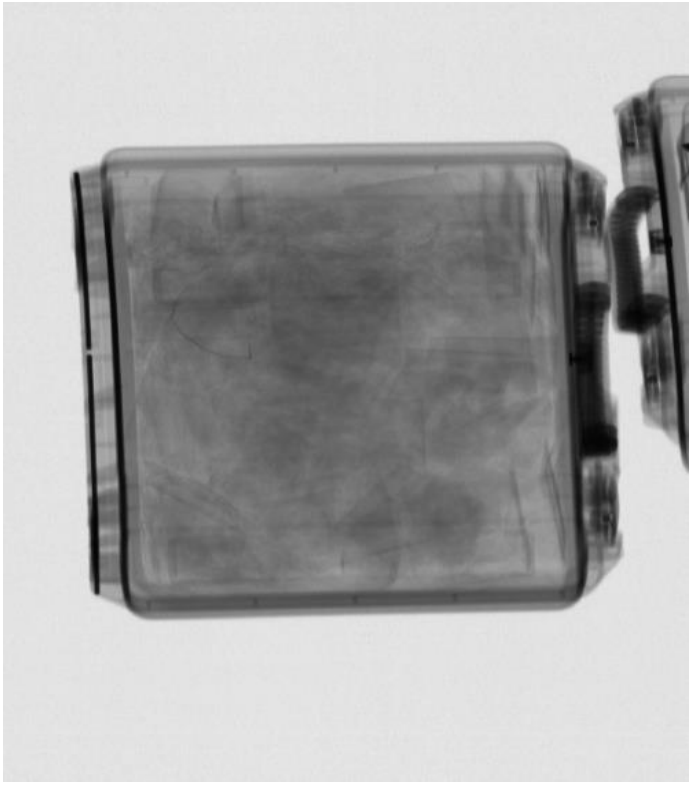


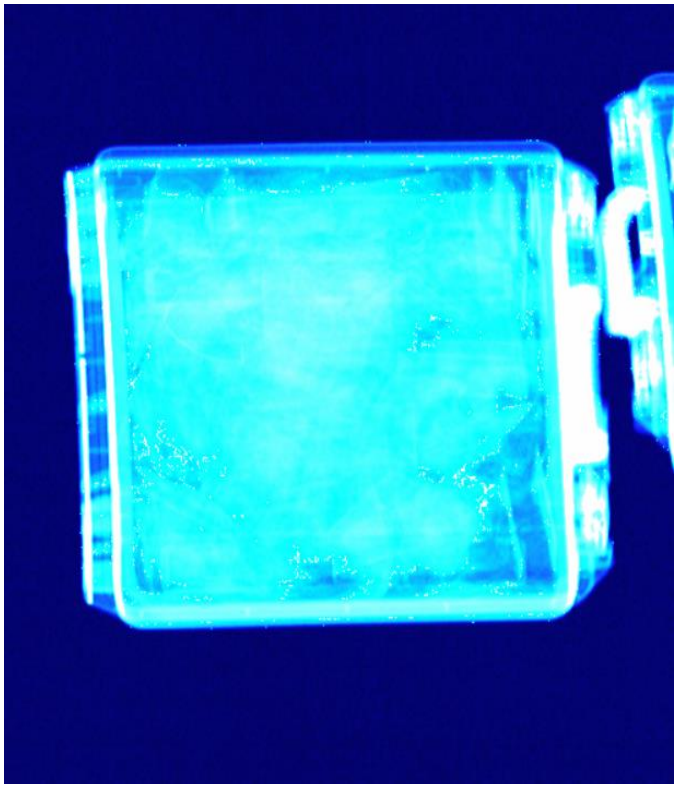
Send your edited HTML to 3d@kbvis.com


```
// create blue->cyan->white spectrum colour table
for (var i = 0; i < 256; i++) {
    if (i < 64) {
        byteTable.push(0);
        byteTable.push(0);
        byteTable.push(i * 4 - 1);
    } else if (i < 128) {
        byteTable.push(0);
        byteTable.push((i - 64) * 4 - 1);
        byteTable.push(255);
    } else if (i < 192) {
        byteTable.push((i - 128) * 3 - 1);
        byteTable.push(255);
        byteTable.push(255);
    } else {
        byteTable.push(255);
        byteTable.push(255);
        byteTable.push(255);
    }
}
```

Follow-up Exercise – Invert the LUT lookup to White => Cyan => Blue spectrum

KB-VIS
We Work In-Depth





Modify only this line in the shader:

```
// get index for looking up colour table
int index = int((imageColour.r)*255.0) + u_lutOffset;
```

Send your edited HTML to 3d@kbvis.com

```
var fragmentShaderSource = `#version 300 es
precision highp float;

uniform sampler2D u_image;
uniform sampler2D u_colourTableTexture;
uniform int u_lutOffset;

// the texCoords passed in from the vertex shader.
in vec2 v_texCoord;

out vec4 color;

void main() {
    // sample the source image (single-channel/grayscale/intensity/luminance)
    vec4 imageColour = texture(u_image, v_texCoord);

    // get index for looking up colour table
    int index = int((1.0-imageColour.r)*255.0) + u_lutOffset;
    index = min(255, max(0, index));
    ivec2 tableTexCoord = ivec2(index, 0);

    // sample (look-up) the colour table to get the final colour
    color = texelFetch(u_colourTableTexture, tableTexCoord, 0);
}
`;
```

Exercise: Streaming Texture – Reverse Scroll Direction (Right to Left)

Send edited HTML to 3d@kbvis.com

```
function updateTexture()
{
    // we want to replace a bit of the texture from the right hand side
    //var column = image.width - (frame * columnsToUpdate) % image.width - columnsToUpdate;
    var column = (frame * columnsToUpdate) % image.width;

    // generate intensities for a grayscale ramp
    var intensity = (frame * columnsToUpdate * 0.05) % 256;
    for (var i = 0; i < image.height * columnsToUpdate; i++) {
        newColumn[i * 4 + 0] = intensity;
        newColumn[i * 4 + 1] = intensity;
        newColumn[i * 4 + 2] = intensity;
        newColumn[i * 4 + 3] = 255;
    }

    ++frame;

    var buffer = new Uint8Array(newColumn);

    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texSubImage2D(gl.TEXTURE_2D, 0, column, 0, columnsToUpdate, image.height,
        gl.RGBA, gl.UNSIGNED_BYTE, buffer);
}
<script id="2d-fragment-shader" type="x-shader/x-fragment">#version 300 es // has to be on first line
    precision mediump float;

    // our texture
    uniform sampler2D u_image;
    uniform sampler2D u_image2;
    uniform vec2 u_textureSize;
    uniform float u_frame;
    uniform float u_columnCount;

    // the texCoords passed in from the vertex shader.
    in vec2 v_texCoord;

    out vec4 fragColor;

    void main() {
        float pixelStep = u_columnCount / u_textureSize.x;
        //fragColor = texture(u_image, vec2(v_texCoord.x - u_frame*pixelStep, v_texCoord.y));
        // right-to-left
        fragColor = texture(u_image, vec2(v_texCoord.x + u_frame*pixelStep, v_texCoord.y));
    }
</script>
```

KB-VIS
We Work In-Depth

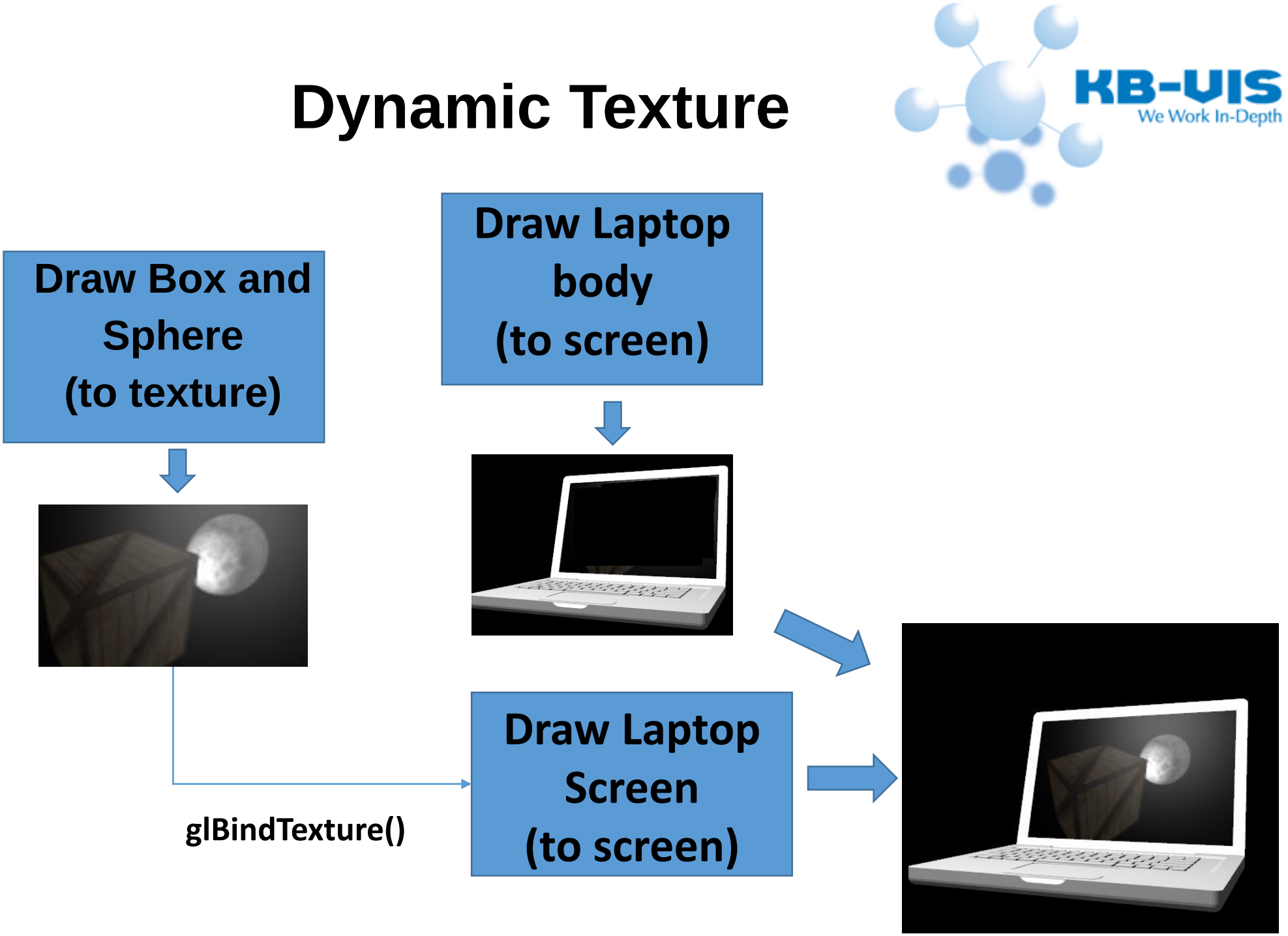
Example:
Dynamic Texture

See examples/dynamictexture

FBOs

- **Framebuffer Objects –** used to render to off-screen targets
- Can be used for post-processing effects
- Render to texture
- Attach images of type color, depth, stencil
- **gl.createFramebuffer**()
- **gl.bindFramebuffer**()

Dynamic Texture



FBO Creation


```
function initTextureFramebuffer() {
    rttFramebuffer = gl.createFramebuffer();
    gl.bindFramebuffer(gl.FRAMEBUFFER, rttFramebuffer);
    rttFramebuffer.width = 512;
    rttFramebuffer.height = 512;

    rttTexture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, rttTexture);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);
    gl.generateMipmap(gl.TEXTURE_2D);

    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, rttFramebuffer.width, rttFramebuffer.height, 0, gl.RGBA,
                                  gl.UNSIGNED_BYTE, null);

    var renderbuffer = gl.createRenderbuffer();
    gl.bindRenderbuffer(gl.RENDERBUFFER, renderbuffer);
    gl.renderbufferStorage(gl.RENDERBUFFER, gl.DEPTH_COMPONENT16, rttFramebuffer.width, rttFramebuffer.height);

    gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, rttTexture, 0);
    gl.framebufferRenderbuffer(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT, gl.RENDERBUFFER, renderbuffer);

    gl.bindTexture(gl.TEXTURE_2D, null);
    gl.bindRenderbuffer(gl.RENDERBUFFER, null);
    gl.bindFramebuffer(gl.FRAMEBUFFER, null);
}
```

- Create a Framebuffer object
- Bind it, so we render to it instead of the default on-screen framebuffer
- Allocate a 2D texture that will receive the rendered output
- Create a renderbuffer with storage for a 16-bit depth buffer
- Attach the texture and depth buffer to the off-screen framebuffer

Render to Texture


```
function drawScene() {
       // render the moon-crate scene to texture (off screen FBO)
    gl.bindFramebuffer(gl.FRAMEBUFFER, rttFramebuffer);
    drawSceneOnLaptopScreen();

    gl.bindFramebuffer(gl.FRAMEBUFFER, null);
            ...

       // draw the laptop object
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, laptopVertexIndexBuffer);
    gl.drawElements(gl.TRIANGLES, laptopVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
            ...
       // draw the laptop screen, texture with the FBO content
    gl.bindBuffer(gl.ARRAY_BUFFER, laptopScreenVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
                                    laptopScreenVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, laptopScreenVertexNormalBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexNormalAttribute,
                                    laptopScreenVertexNormalBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ARRAY_BUFFER, laptopScreenVertexTextureCoordBuffer);
    gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
                                    laptopScreenVertexTextureCoordBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, rttTexture);
    gl.uniform1i(shaderProgram.samplerUniform, 0);

    setMatrixUniforms();
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, laptopScreenVertexPositionBuffer.numItems);

    mvPopMatrix();
}
```

DAY 11

Transparency & Blending

- **Blending** colors to make objects appear translucent

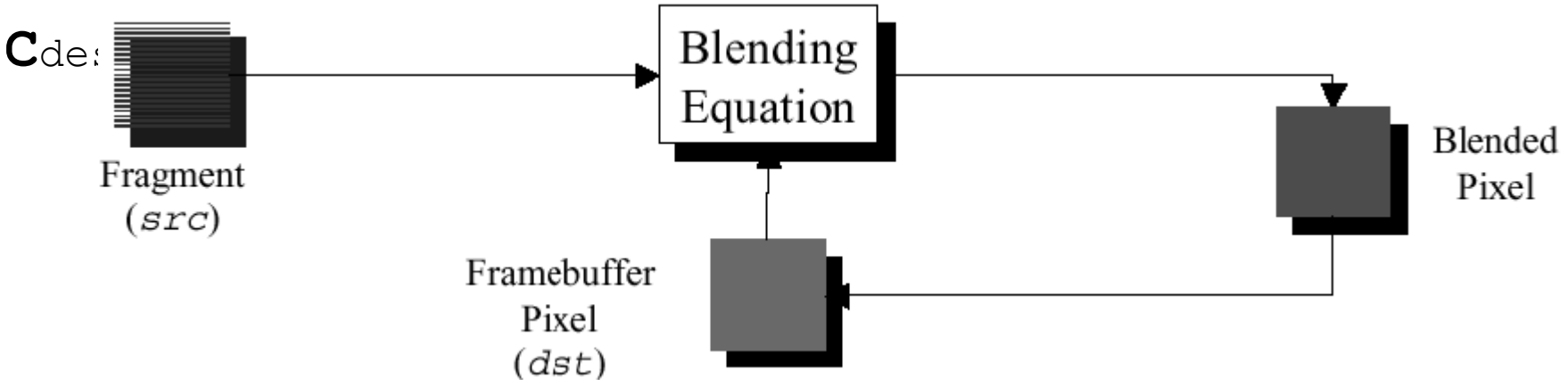
glEnable(GL_BLEND)

- **Blending function** specifies how color values from a

source and a destination are combined:

glBlendFunc(GLenum sfactor, GLenum dfactor)

- color values of incoming fragment (*source*) are combined with the color values of the corresponding currently stored pixel (*destination*):

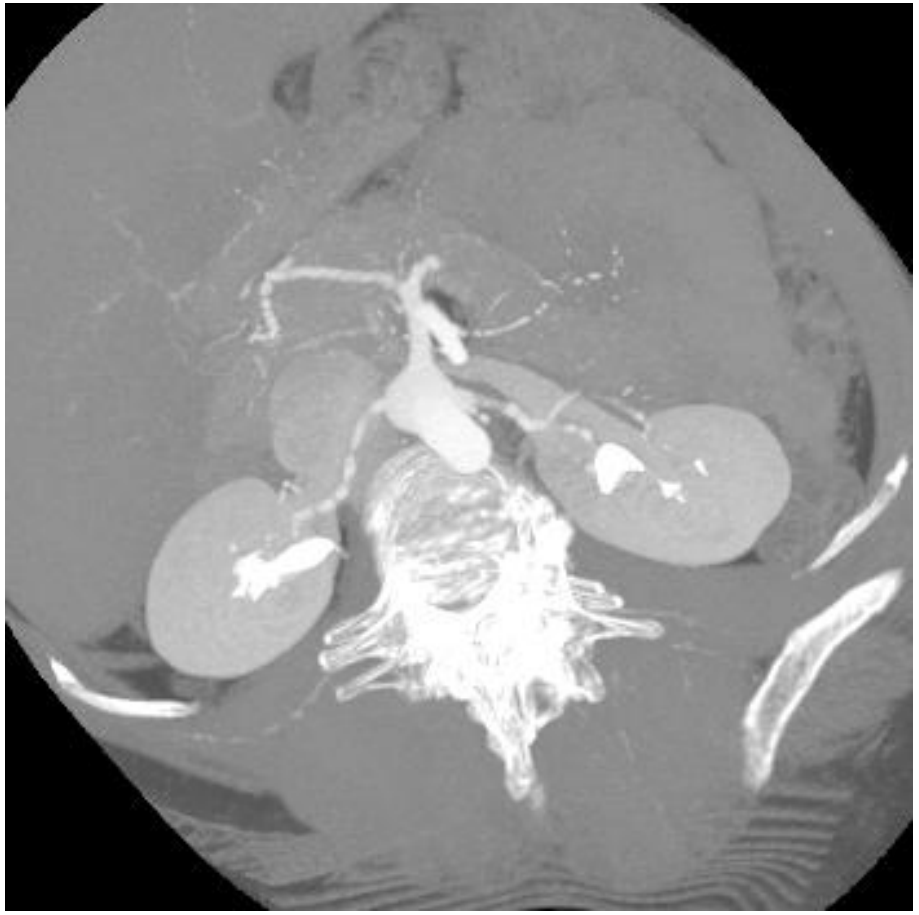


Blend Function

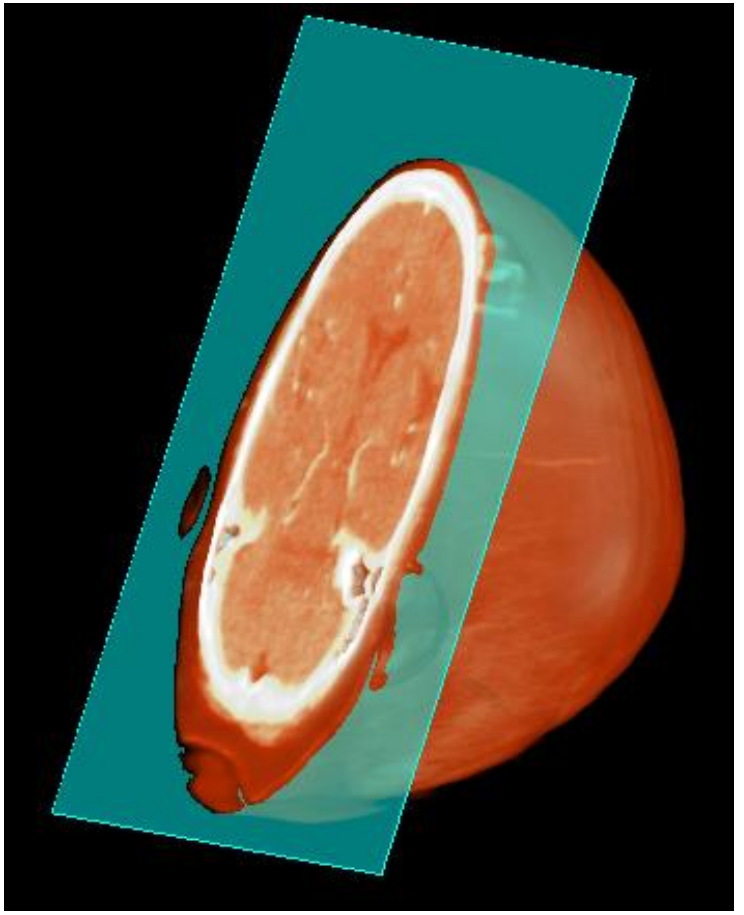
KB-VIS
We Work In-Depth

Constant	Relevant Factor	Computed Blend Factor
GL_ZERO	source or destination	(0, 0, 0, 0)
GL_ONE	source or destination	(1, 1, 1, 1)
GL_DST_COLOR	source	(Rd, Gd, Bd, Ad)
GL_SRC_COLOR	destination	(Rs, Gs, Bs, As)
GL_ONE_MINUS_DST_COLOR	source	(1, 1, 1, 1)-(Rd, Gd, Bd, Ad)
GL_ONE_MINUS_SRC_COLOR	destination	(1, 1, 1, 1)-(Rs, Gs, Bs, As)
GL_SRC_ALPHA	source or destination	(As, As, As, As)
GL_ONE_MINUS_SRC_ALPHA	source or destination	(1, 1, 1, 1)-(As, As, As, As)
GL_DST_ALPHA	source or destination	(Ad, Ad, Ad, Ad)
GL_ONE_MINUS_DST_ALPHA	source or destination	(1, 1, 1, 1)-(Ad, Ad, Ad, Ad)
GL_SRC_ALPHA_SATURATE	source	(f, f, f, 1); f=min(As, 1-Ad)

Blend Function



glBlendEquation(GL_MAX)

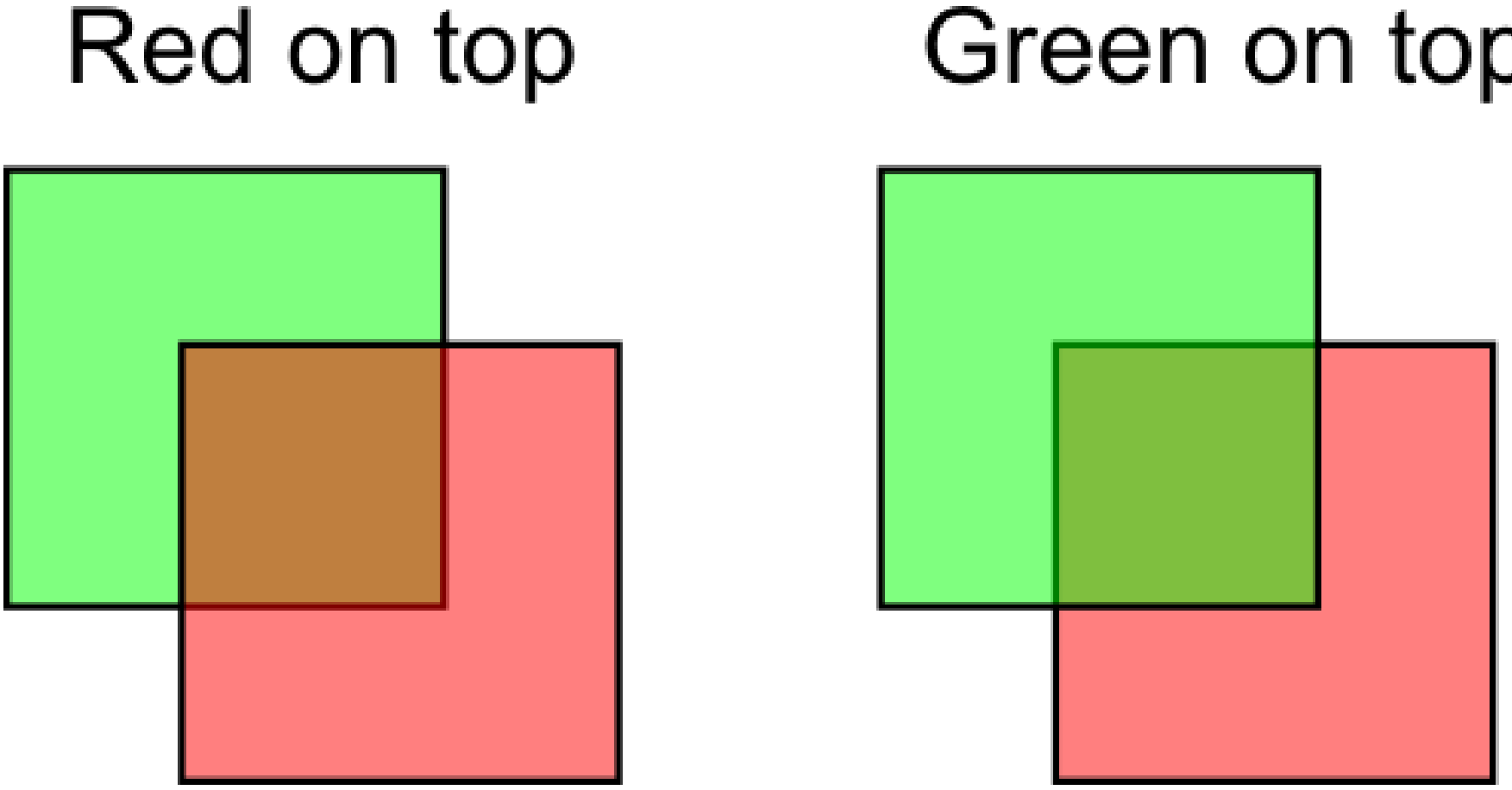


glBlendFunc(
GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA)

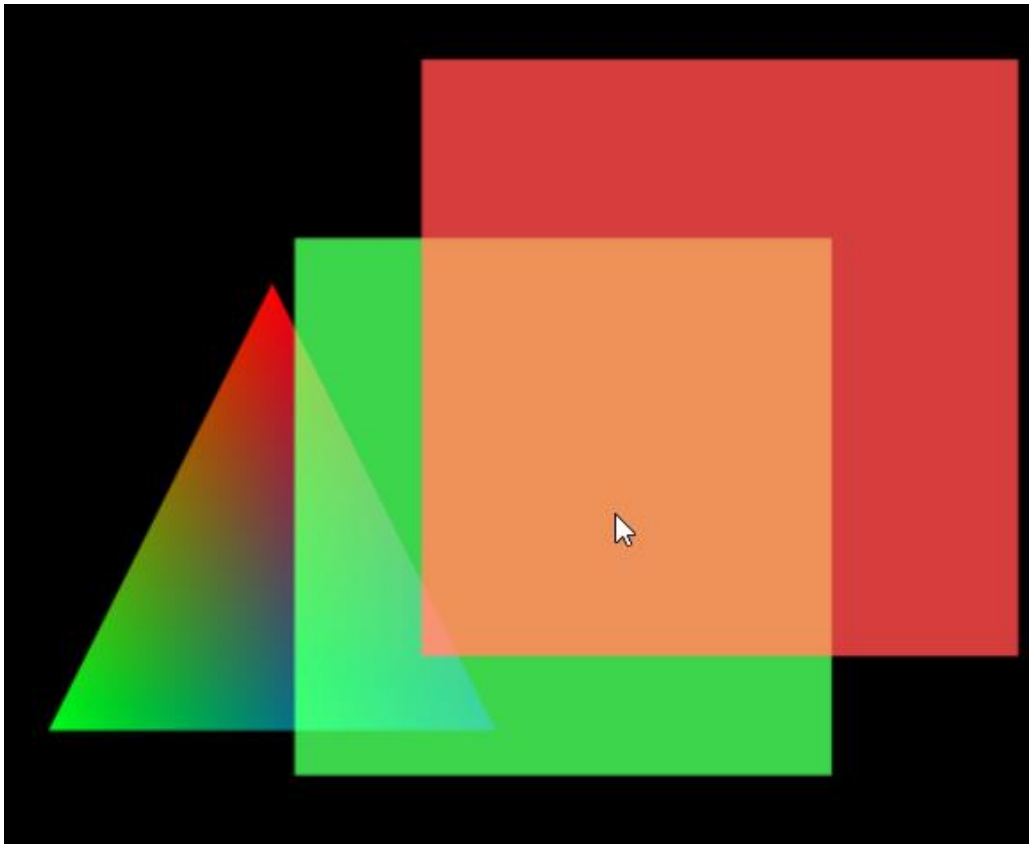
Blend Operations

- **glBlendFunc**(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
  - known as the "over" operator, used to blend or *composite* a fragment over the one behind it
  - Dependent on rendering order – objects must be rendered in back-to-front order (w.r.t. eye)
  - Commonly used in rendering translucent or semi-transparent objects, especially in volume rendering, e.g. medical volume visualization
- **glBlendEquation**(GL_MAX)
  - retains the fragment with maximum opacity (alpha)
  - Independent of render order
  - Commonly used in medical visualization to view high-intensity areas (bone, vasculature with contrast agent)

“Over” Operator



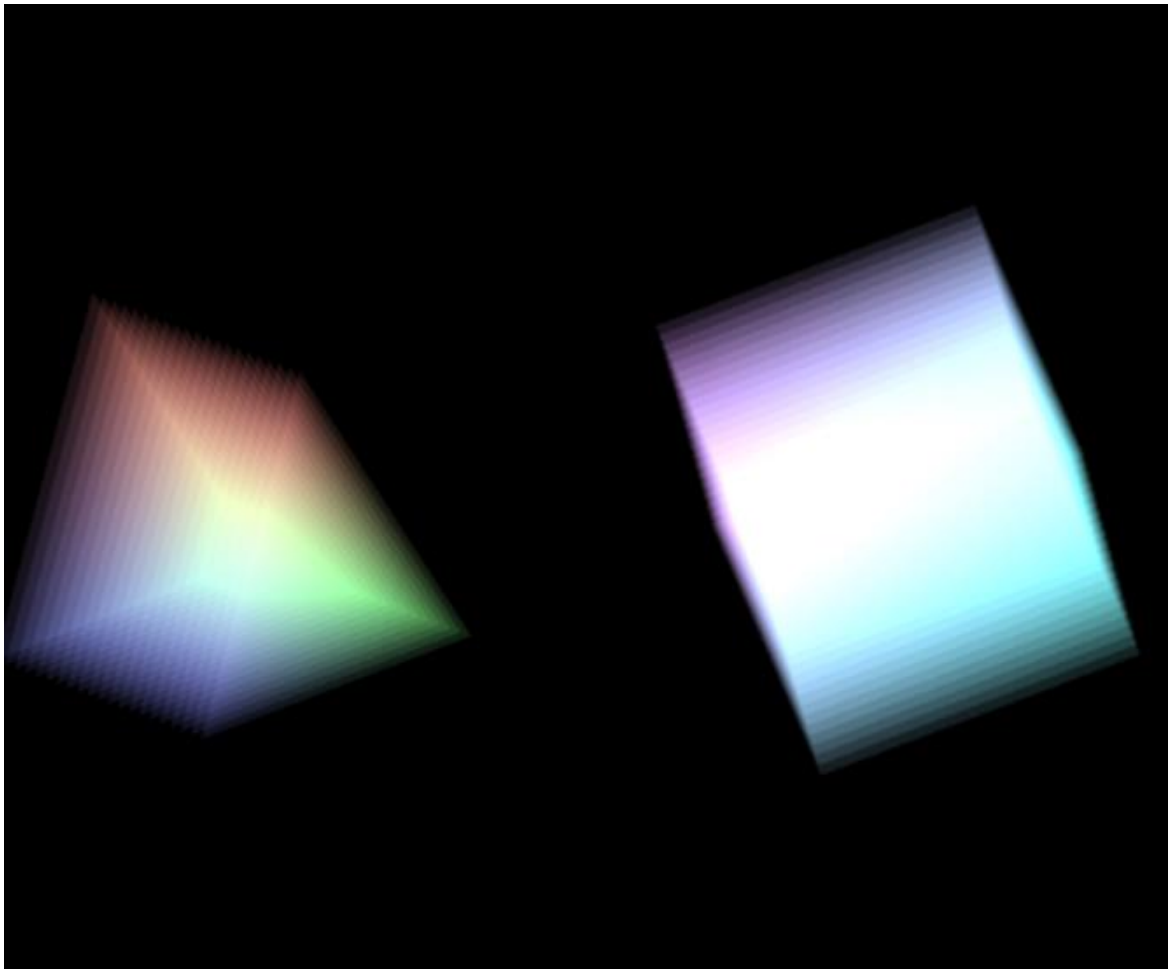
Example: Blending



See: example10-transparency-and-blending

```
// draw transparent squares in front of the triangle

mat4.translate(mvMatrix, [1.25, 0.0, 0.0]);
setMatrixUniforms();
// use offset to skip first three entries of the triangle (7 floats per vertex)
offset = 3 * 7 * 4;
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, 3, gl.FLOAT, false, stride, offset);
// color values start 12 bytes after position (3 floats)
//offset += 12;
//gl.vertexAttribPointer(shaderProgram.vertexColorAttribute, 4, gl.FLOAT, false, stride, offset);
// use transparent green instead of vertex colours
gl.disableVertexAttribArray(shaderProgram.vertexColorAttribute);
gl.vertexAttrib4f(shaderProgram.vertexColorAttribute, 0.0, 1.0, 0.0, 0.6);
gl.enable(gl.BLEND);
gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);

mat4.translate(mvMatrix, [0.5, 0.5, 0.5]);
setMatrixUniforms();
// use transparent red
gl.disableVertexAttribArray(shaderProgram.vertexColorAttribute);
gl.vertexAttrib4f(shaderProgram.vertexColorAttribute, 1.0, 0.0, 0.0, 0.6);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);

gl.disable(gl.BLEND);
```

Example: Transparent Stack



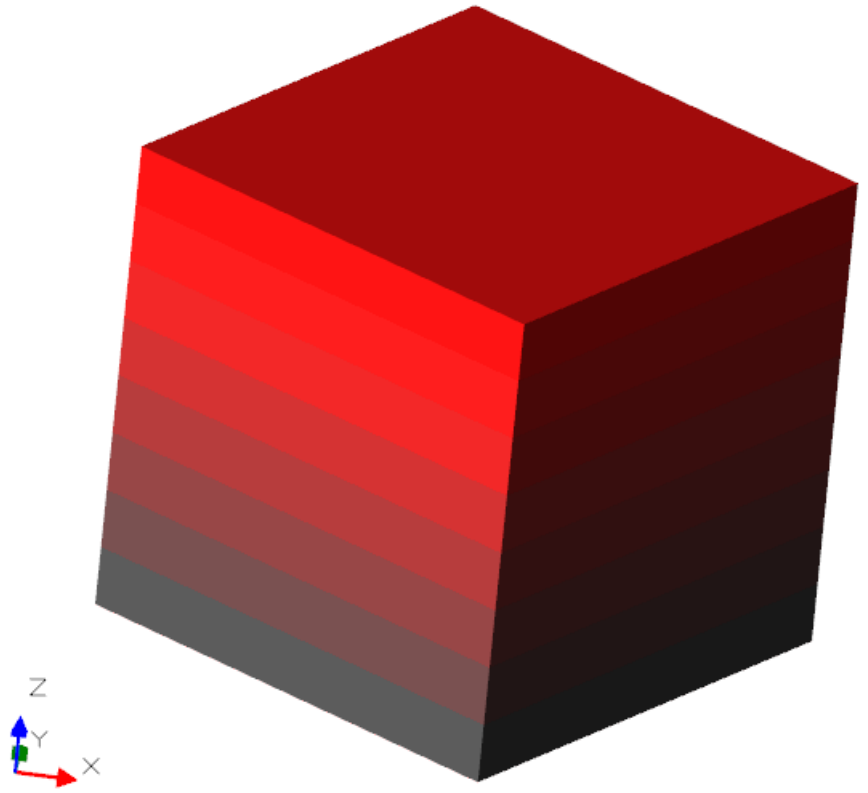
See: example10-transparency-and-blending

Transparent Stack

- Approximation to Volume Rendering
- Draw transparent layers on top of one another
- Use “over” operator for compositing fragment colors
- Accumulation is highest in regions of maximum overlap (more fragments are blended)
- Layers must be drawn in depth-order to get meaningful result.

```
// draw transparent stack, by repeatedly drawing with displacement (translation of MV matrix)
for (var layer = -1.0; layer < 1.0; layer += 0.1) {

    mat4.identity(mvMatrix);
    mat4.translate(mvMatrix, [-2, 0.0, -7.0]);
    mat4.rotate(mvMatrix, degToRad(initialRotation), [1, 1, 0.5]);
    mvPushMatrix();
    mat4.rotate(mvMatrix, degToRad(rTri), [0, 1, 0]);
    mat4.translate(mvMatrix, [0.0, 0.0, layer]);
    setMatrixUniforms();
    mvPopMatrix();

    // draw triangle with displacement along stack
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute, triangleVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);
    gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);

    mat4.identity(mvMatrix);
    mat4.translate(mvMatrix, [2.0, 0.0, -7.0]);
    mat4.rotate(mvMatrix, degToRad(initialRotation), [1, 0, 1]);
    mvPushMatrix();
    mat4.rotate(mvMatrix, degToRad(rSquare), [1, 0, 0]);
    mat4.translate(mvMatrix, [0.0, 0.0, layer]);
    setMatrixUniforms();
    mvPopMatrix();

    // draw square with displacement along stack
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexColorAttribute, squareVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, squareVertexPositionBuffer.numItems);
}
```

Example: 3D Texture



See: example11-3d-texture


```
void glTexImage3D( GLenum target,
                   GLint level,
                   GLint internalFormat,
                   GLsizei width,
                   GLsizei height,
                   GLsizei depth,
                   GLint border,
                   GLenum format,
                   GLenum type,
                   const GLvoid * data);
```

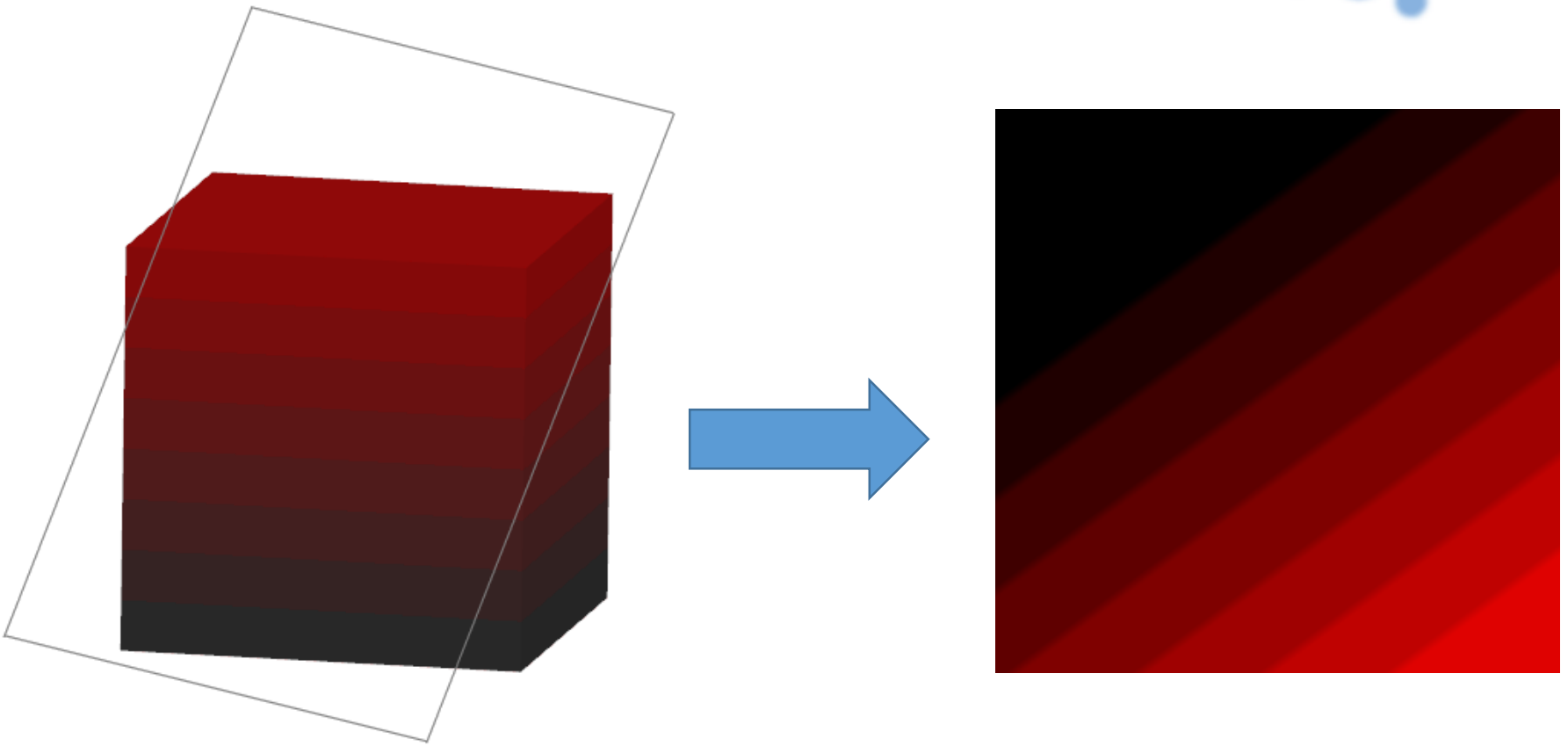
```
void glTexSubImage3D( GLenum target,
                      GLint level,
                      GLint xoffset,
                      GLint yoffset,
                      GLint zoffset,
                      GLsizei width,
                      GLsizei height,
                      GLsizei depth,
                      GLenum format,
                      GLenum type,
                      const GLvoid * data);
```

Fragment Shader:

```
gvec texture(gsampler sampler, vec texCoord[, float bias]);
```

texCoord must be vec3

Sampling the 3D Texture



- Use 3D texture coordinates (s,t,p) as vertex attributes
- In **glTexParameter** use TEXTURE_WRAP_R for third texture dimension

Creating the 3D Texture


```
var texture = gl.createTexture();
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_3D, texture);
gl.texParameteri(gl.TEXTURE_3D, gl.TEXTURE_BASE_LEVEL, 0);
gl.texParameteri(gl.TEXTURE_3D, gl.TEXTURE_MAX_LEVEL, Math.log2(SIZE));
gl.texParameteri(gl.TEXTURE_3D, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_LINEAR);
gl.texParameteri(gl.TEXTURE_3D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
gl.texParameteri(gl.TEXTURE_3D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
gl.texParameteri(gl.TEXTURE_3D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
gl.texParameteri(gl.TEXTURE_3D, gl.TEXTURE_WRAP_R, gl.CLAMP_TO_EDGE);

gl.texImage3D(
    gl.TEXTURE_3D,  // target
    0,              // level
    gl.R8,        // internalformat
    SIZE,           // width
    SIZE,           // height
    SIZE,           // depth
    0,              // border
    gl.RED,         // format
    gl.UNSIGNED_BYTE,       // type
    data            // pixel
    );

gl.generateMipmap(gl.TEXTURE_3D);
```

Using the 3D Texture


```
// bind our 3d texture to texture unit 0
gl.uniform1i(uniformVolumeTextureLocation, 0);
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_3D, texture);

gl.bindVertexArray(vertexArray);
// pass current texture rotation matrix
gl.uniformMatrix4fv(uniformTextureMatrixLocation, false, yawPitchRollMatrix);
// draw textured quad (two triangles)
gl.drawArrays(gl.TRIANGLES, 0, 6);
```

Transforming the 3D Texture Coordinate


```
<script id="vs" type="x-shader/x-vertex">
    #version 300 es
    precision highp float;
    precision highp int;

    layout(location = 0) in vec2 position;
    layout(location = 1) in vec2 in_texcoord;

    // Output 3D texture coordinate after transformation
    out vec3 v_texcoord;

    // Matrix to transform the texture coordinates into 3D space
    uniform mat4 orientation;

    void main()
    {
        // Multiply the texture coordinate by the transformation
        // matrix to place it into 3D space
        v_texcoord = (orientation * vec4(in_texcoord - vec2(0.5, 0.5), 0.5, 1.0)).stp + vec3(0.5,0.5,0.5);
        gl_Position = vec4(position, 0.0, 1.0);
    }

</script>
```

Sampling the 3D Texture


```
<script id="fs" type="x-shader/x-fragment">
    #version 300 es

    precision highp float;
    precision highp int;
    precision highp sampler3D;

    uniform sampler3D volumeTexture;

    in vec3 v_texcoord;

    out vec4 color;

    void main()
    {
        color = texture(volumeTexture, v_texcoord);
    }
</script>
```

Layout Qualifiers

Vertex shader attribute index

Vertex shader inputs can specify the attribute index that the particular input uses. This is done with this syntax:

```
layout(location = attribute index) in vec3 position;
```

With this syntax, you can forgo the use of `glBindAttribLocation` entirely. If you try to combine the two and they conflict, the layout qualifier always wins.

Attributes that take up multiple attribute slots will be given a sequential block of that number of attributes in order starting with the given attribute. For example:

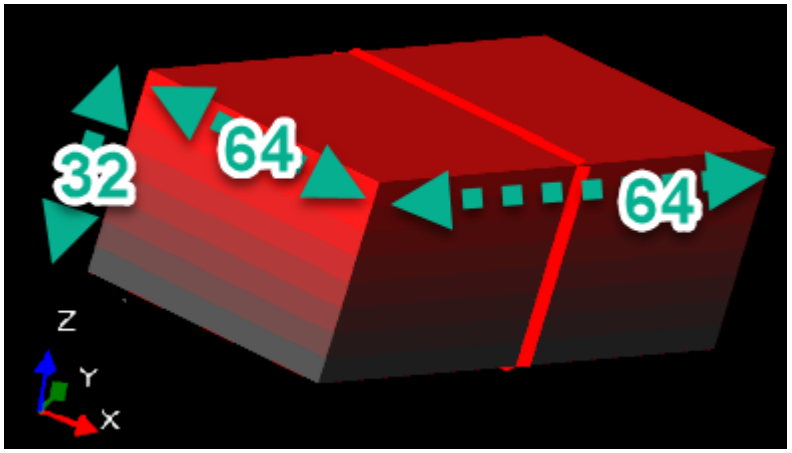
```
layout(location = 2) in vec3 values[4];
```

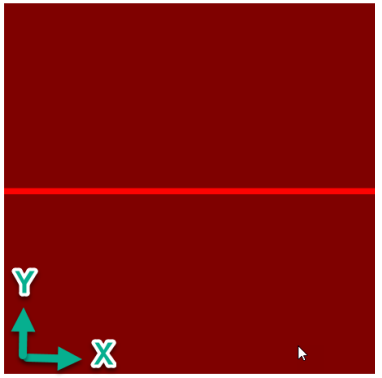
This will allocate the attribute indices 2, 3, 4, and 5.

KB-VIS
We Work In-Depth

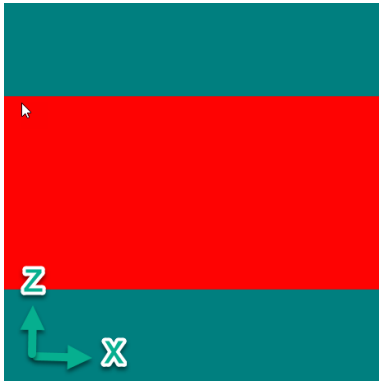
DAY 12

Simplified Example – Standard Slice Planes

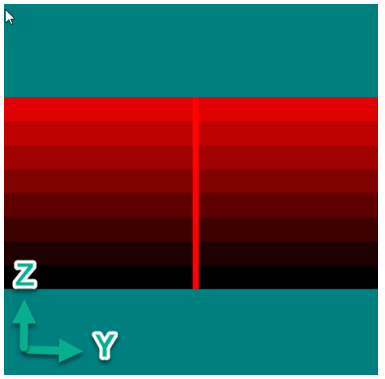




Axial (XY)



Coronal (XZ)



Sagittal (YZ)

```
<script id="vs" type="x-shader/x-vertex">
    #version 300 es
    precision highp float;
    precision highp int;

    layout(location = 0) in vec2 position;
    layout(location = 1) in vec3 in_texcoord;

    // Output 3D texture coordinate after transformation
    out vec3 v_texcoord;

    void main()
    {
        // Multiply the texture coordinate by the transformation
        // matrix to place it into 3D space
        v_texcoord = in_texcoord;
        gl_Position = vec4(position, 0.0, 1.0);
    }
</script>
```

```
<script id="fs" type="x-shader/x-fragment">
    #version 300 es

    precision highp float;
    precision highp int;
    precision highp sampler3D;

    uniform sampler3D volumeTexture;

    in vec3 v_texcoord;

    out vec4 color;

    void main()
    {
        // clamp to border colour ( or )
        //if(v_texcoord.p < 0.0 || v_texcoord.p > 1.0)
            //discard; // don't draw fragment (similar to setting color to (0,0,0,0))
        if(v_texcoord.p < 0.0 || v_texcoord.p > 1.0)
            color = vec4(0.0,0.5,0.5,1.0); //cyan
        else
            color = texture(volumeTexture, v_texcoord);
    }
</script>
```

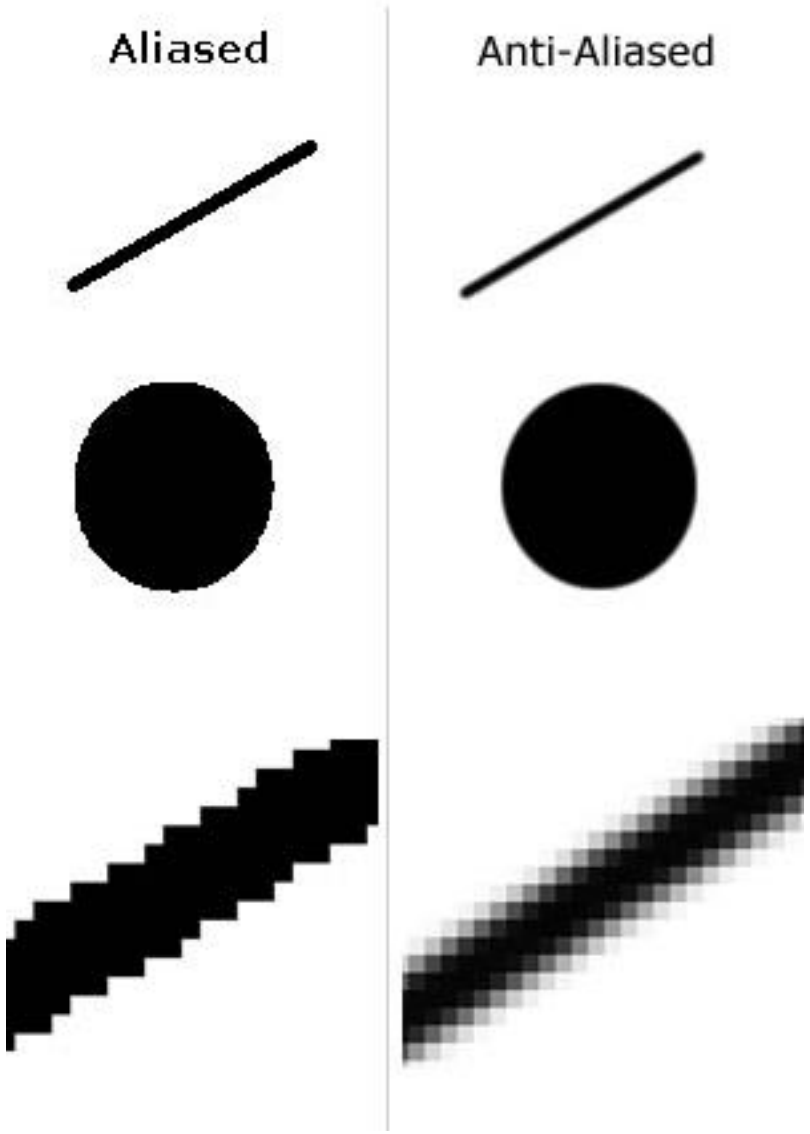
Standard Planes


```
 // axial plane (XY)
 var texCoords = new Float32Array([
     0.0, 1.0, 0.5,
     1.0, 1.0, 0.5,
     1.0, 0.0, 0.5,
     1.0, 0.0, 0.5,
     0.0, 0.0, 0.5,
     0.0, 1.0, 0.5
 ]);

 // coronal plane (XZ)
 var texCoords = new Float32Array([
     0.0, 0.5, 1.0 + texScaleOffset,
     1.0, 0.5, 1.0 + texScaleOffset,
     1.0, 0.5, 0.0 - texScaleOffset,
     1.0, 0.5, 0.0 - texScaleOffset,
     0.0, 0.5, 0.0 - texScaleOffset,
     0.0, 0.5, 1.0 + texScaleOffset,
 ]);
// sagittal plane (YZ)
var texCoords = new Float32Array([
     0.5, 0.0, 0.0 - texScaleOffset,
     0.5, 1.0, 0.0 - texScaleOffset,
     0.5, 1.0, 1.0 + texScaleOffset,
     0.5, 1.0, 1.0 + texScaleOffset,
     0.5, 0.0, 1.0 + texScaleOffset,
     0.5, 0.0, 0.0 - texScaleOffset
]);
```

Suggested Exercises

- Add mouse move event handler to increment/decrement offset in texture coordinates (s,t,p) and observe the sampled result (textured quad)
- Fill the byte array used for the volume texture with different patterns (chequered, intensity function etc.) and observe the slicing behaviour

Antialiasing

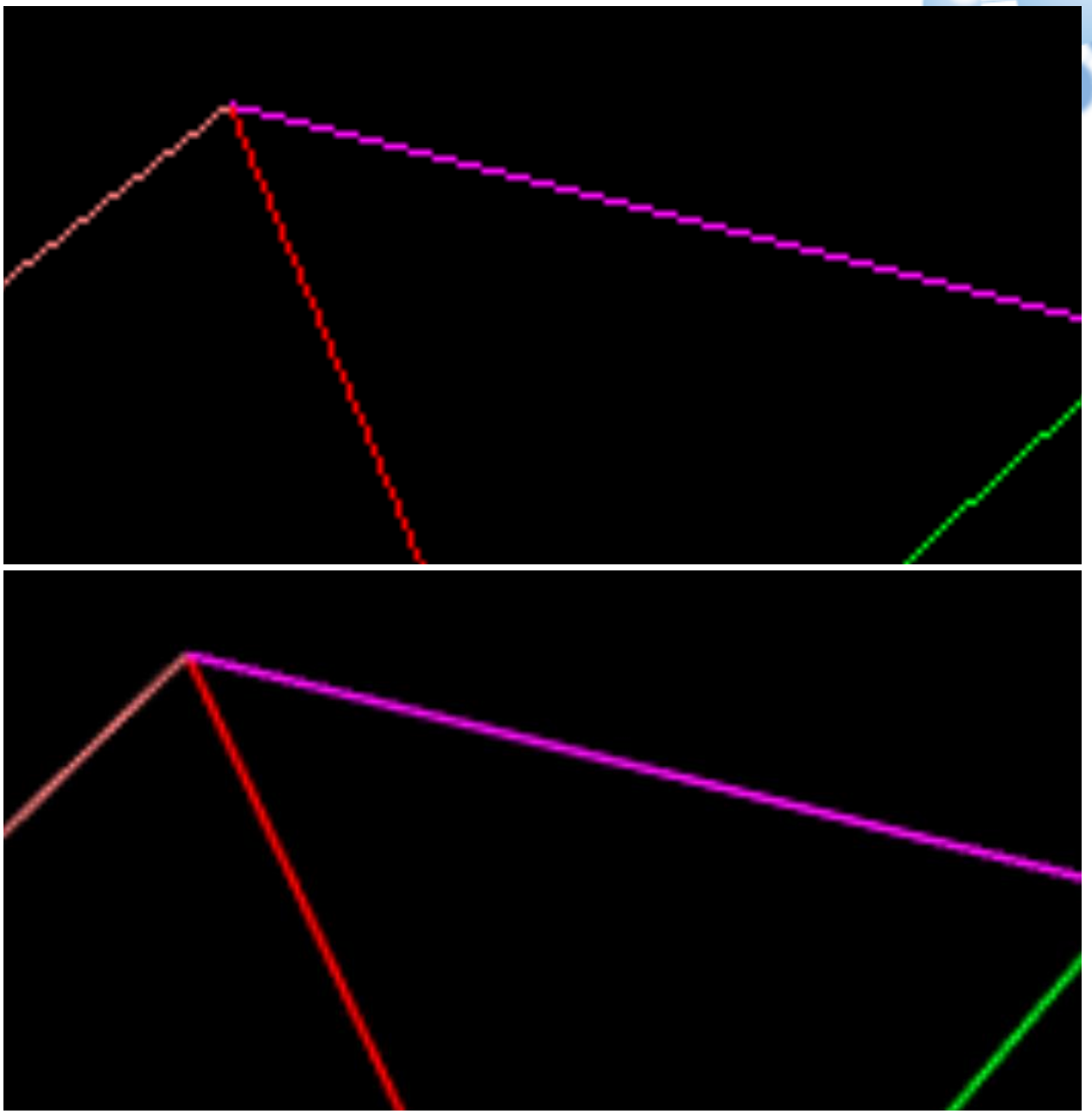


Old API:

```
glEnable(GL_BLEND);

glEnable(GL_POINT_SMOOTH);

glEnable(GL_LINE_SMOOTH);

glEnable(GL_POLYGON_SMOOTH);

glHint(GL_POLYGON_SMOOTH_HINT,
                    GL_NICEST);
```

- ***Not very effective***
- ***"Seams" visible between triangles***
- ***This has been replaced by Multisample Antialiasing (MSAA)***
- ***Just request an anti-aliased WebGL context***

Example: Antialiasing

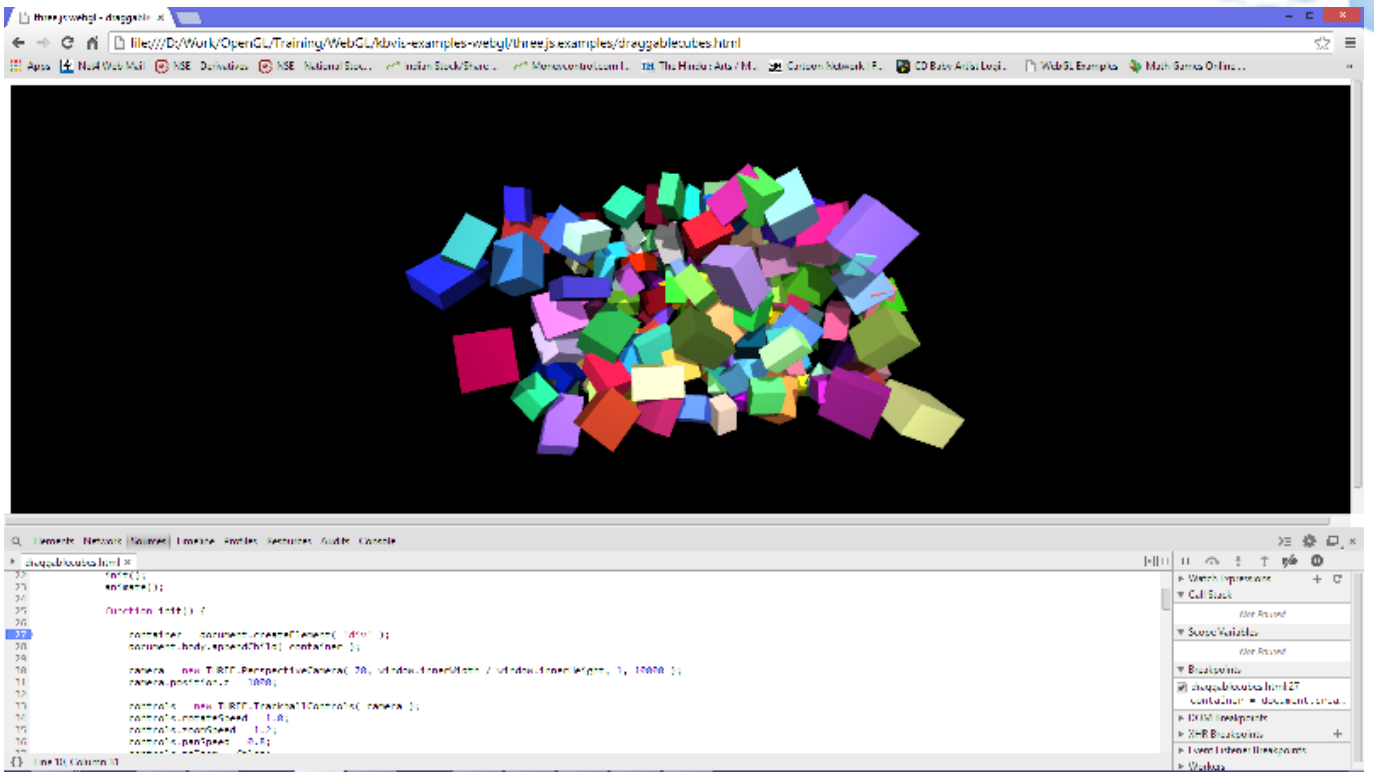


See: example12-multisample


```
function initGL(canvas) {
    try {
        gl = canvas.getContext("webgl", {antialias:true});
        gl.viewportWidth = canvas.width;
        gl.viewportHeight = canvas.height;
    } catch (e) {
    }
    if (!gl) {
        alert("Could not initialise WebGL, sorry :-(");
    }
}
```

Chrome Developer Tools

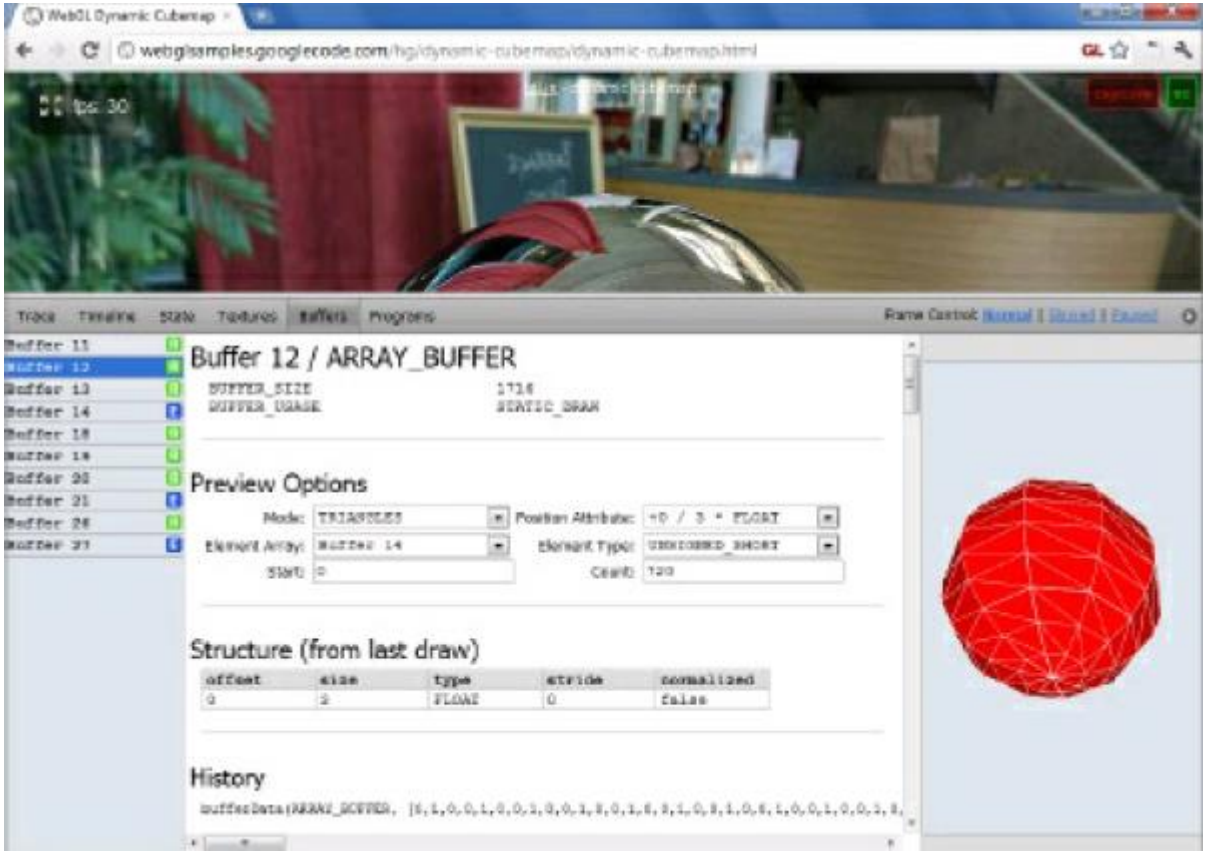
KB-VIS
We Work In-Depth



Tool Panels:

- Elements
- Resources
- Network
- Scripts
- Timeline
- Profiles
- Audits
- Console

WebGL Inspector



Tool Panels:

- Trace
- Timeline
- State
- Textures
- Buffers
- Programs

[Dynamic Cubemap Demo]

Recent Trends

- **OpenGL 3.0-3.3**
  - Geometry Shader
  - Core and Compatibility Profiles
  - Core Profile excludes deprecated features:
    - Fixed-function vertex and fragment processing
    - Direct-mode rendering, using glBegin and glEnd
    - Display lists
    - Indexed-color rendering targets
- **OpenGL 4.0-4.6**
  - Tessellation, Compute Shaders, Shader Atomics
  - Improved compatibility with OpenGL-ES
  - Sparse Textures, Bindless API
  - Increased Double Precision support
- **OpenGL-ES 3.0-3.2**
  - Improved texturing, 32-bit shader precision
- **WebGL 2** – based on OpenGL-ES 3.0

- New generation API
- Direct, low-level GPU control
- Faster performance, lower overhead, less latency
- Portable - Cloud, desktop, console, mobile and embedded
- One unified API framework for desktop, mobile, console, and embedded
- Improved Multi-threading support
- Command Buffers
- https://www.khronos.org/vulkan-overview.pdf

OpenCL

- Framework for executing programs on heterogeneous platforms:
  - Central processing units (CPUs)
  - Graphics processing units (GPUs)
  - Digital signal processors (DSPs)
  - Field-programmable gate arrays (FPGAs)
- OpenCL runs “kernels” on available “compute devices”
- Kernel execution runs on many “processing elements” (PEs) in parallel
- Programming language used to write compute kernels is “OpenCL C” and is based on C99
- C++ Bindings are available

Memory Hierarchy

- OpenCL defines a four-level memory hierarchy for the compute device
- Global memory: shared by all processing elements, but has high access latency
- Read-only (constant) memory: smaller, low latency, writable by host CPU but not compute devices;
- Local memory: shared by a group of processing elements;
- Per-element private memory (registers)

OpenCL Programming

- Program compilation and kernel objects
- Managing Buffers
- Kernel Programming and Design
- Kernel Execution
- Kernel Synchronization
- Performance Tuning
  - Occupancy
  - Memory Coalescing
  - Bank Conflicts

Example: matrix-vector multiplication


```
// Multiplies A*x, leaving the result in y.
// A is a row-major matrix, meaning the (i,j) element is at
A[i*ncols+j].
__kernel void matvec(__global const float *A, __global const
float *x,
                     uint ncols, __global float *y)
{
    size_t i = get_global_id(0);              // Global id,
used as the row index.
    __global float const *a = &A[i*ncols];    // Pointer to
the i'th row.
    float sum = 0.f;                          // Accumulator
for dot product.
    for (size_t j = 0; j < ncols; j++) {
        sum += a[j] * x[j];
    }
    y[i] = sum;
}
```

Example - CLAHE


```
__kernel void ComputeHistogram(__read_only image2d_t bmp,
    __global float * partHist,
    __constant int * imgHeight,
    __constant int * NLevels)

{
    int x = get_global_id(0); // current column in image
    int h = imgHeight[0]; // row count in image
    int N = NLevels[0]; //Intensity levels

    int2 coord = (int2)(x, 0);
    uint4 pix;
    int R, G, B;
    float4 hsl;  // Hue-Saturation-Luminance representation of the RGB colour

    int localHist[1024];
    for (int i = 0; i < N; i++) localHist[i] = 0.0f;
```

Histogram Kernel – Initialise local histogram

Example - CLAHE


```
for (int y = 0; y < h; y++) // loop through the column
{
    coord.y = y;
    pix = read_imageui(bmp, smp, coord);

    B = (int)pix.x;
    G = (int)pix.y;
    R = (int)pix.z;

    hsl = RGBtoHSL(DivBy255[R], DivBy255[G], DivBy255[B]);
    localHist[(int)((N - 1) * hsl.z)]++;
}

for (int i = 0; i < N; i++) partHist[i + N*x] = (float)localHist[i];
```

Histogram Kernel – Compute partial histogram

Example - CLAHE


```
kernel void ConsolidateHist(__global const float* PartialHistograms,
    __global       float* Histograms,
    __constant      int* Height,
    __constant      int* NLevels)

{
    int N = NLevels[0];
    int h = Height[0];
    int i = get_global_id(0);

    float val = 0;
    int yN = 0;
    for (int y = 0; y < h; y++)
    {
        val += PartialHistograms[i + yN];
        yN += N;
    }

    Histograms[i] = val;
}
```

Histogram Kernel – Combine partial histograms

Example - CLAHE


```
__kernel void PerformNormalization(read_only  image2d_t   bmp,
    write_only image2d_t   bmpNew,
    __global const float * histLuminance,
    __constant     int *   NLevels)
{
    int N = NLevels[0];

    int2 coord = (int2)(get_global_id(0), get_global_id(1));
    uint4 pix = read_imageui(bmp, smp, coord);

    float B = (float)pix.x;
    float G = (float)pix.y;
    float R = (float)pix.z;

    float4 hsl = RGBtoHSL(R * 0.00392156862745098f, G * 0.00392156862745098f, B * 0.00392156862745098f);
    hsl.z = histLuminance[(int)((N - 1) * hsl.z)];

    float4 rgb = HSLtoRGB(hsl.x, hsl.y, hsl.z);
    rgb = clamp(rgb, 0.0f, 1.0f);
    rgb *= 255.0f;

    pix = (uint4)((uint)rgb.z, (uint)rgb.y, (uint)rgb.x, (uint)rgb.w);
    write_imageui(bmpNew, coord, pix);
}
```

CLAHE Kernel – Equalise Histogram

Resources

- OpenGL SuperBible, Sellers, Wright, Haemel
- OpenGL Shading Language, 3rd Edition, Randi Rost
- www.opengl.org/documentation/specs/, Official OpenGL Documentation
- OpenGL FAQ
- OpenGL Common Mistakes
- Computer Graphics: Principles and Practice, Foley, van Dam, Hughes
- Mathematical Elements of Computer Graphics, Rogers, Adams
- CS 410: Introduction to Computer Graphics, Colorado State University
- WebGL: Up and Running, Tony Parisi, O'Reilly Press
- WebGL Beginner's Guide, Diego Cantor and Brandon Jones
- http://www.songho.ca/opengl/gl_pbo.html (PBO Tutorial)
- Introduction to Accelerated Computing

Thank You !