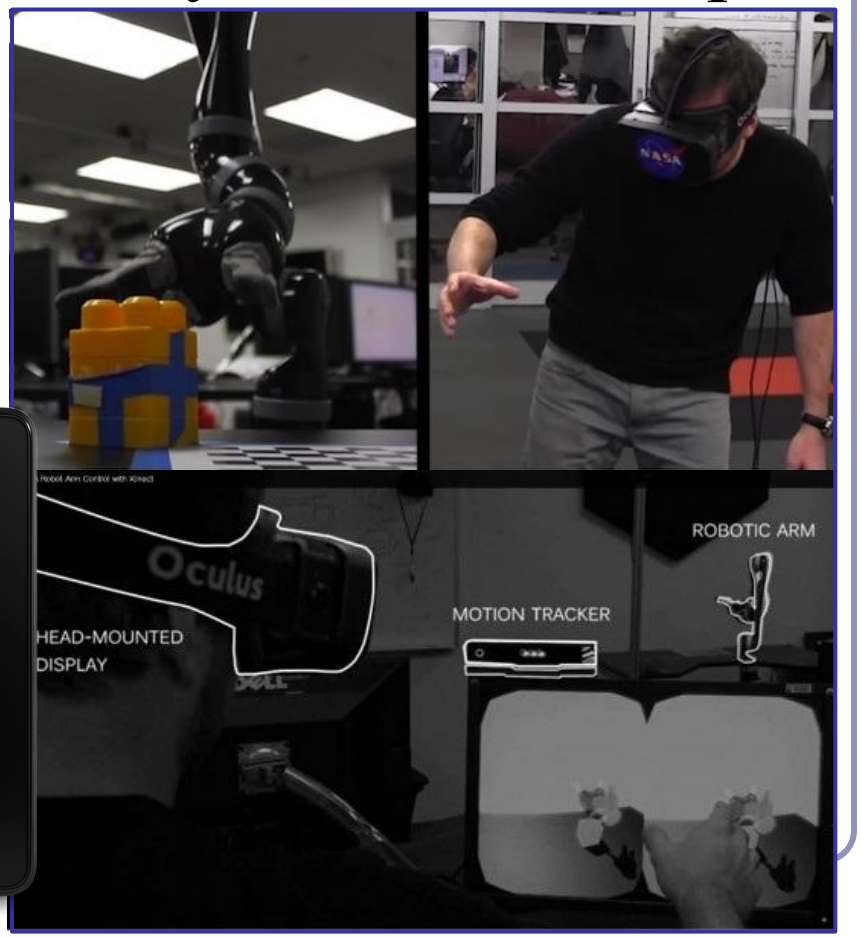
Advanced Graphics

Beyond the desktop



Alex Benton, University of Cambridge – A.Benton@damtp.cam.ac.uk

Supported in part by Google UK, Ltd

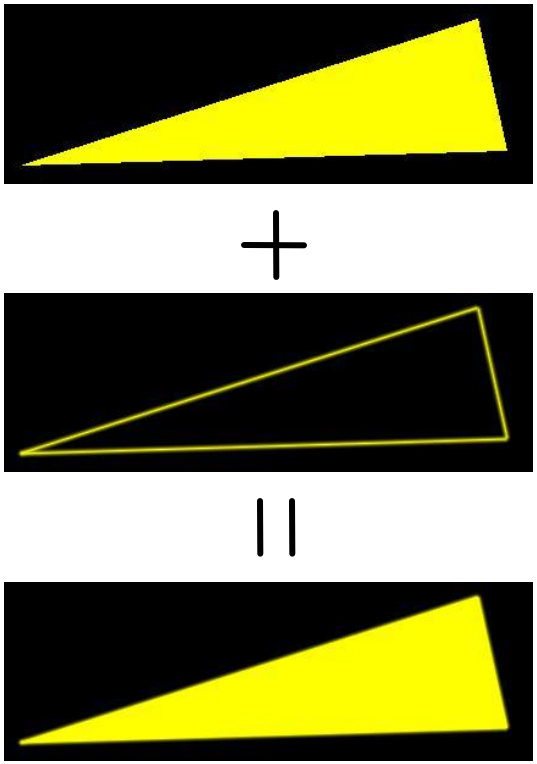
Antialiasing on the GPU

Hardware antialiasing can dramatically improve image quality.

- The naïve approach is to supersample the image
- This is easier in shaders than it is in standard software
- But it really just postpones the problem.

Several GPU-based antialiasing solutions have been found.

- Eric Chan published an elegant polygon-based antialiasing approach in 2004 which uses the GPU to prefilter the edges of a model and then blends the filtered edges into the original polygonal surface. (See figures at right.)



Antialiasing on the GPU

One clever form of antialiasing is *adaptive analytic prefiltering*.

- The precision with which an edge is rendered to the screen is dynamically refined based on the rate at which the function defining the edge is changing with respect to the surrounding pixels on the screen.

This is supported in the shader language by the methods `dFdx(F)` and `dFdy(F)`.

- These methods return the derivative with respect to X and Y of some variable F.
- These are commonly used in choosing the filter width for antialiasing procedural textures.

(A) Jagged lines visible in the box function of the procedural stripe texture
(B) Fixed-width averaging blends adjacent samples in texture space; aliasing still occurs at the top, where adjacency in texture space does not align with adjacency in pixel space.
(C) Adaptive analytic prefiltering smoothly samples both areas.
Image source: Figure 17.4, p. 440, *OpenGL Shading Language, Second Edition*, Randi Rost, Addison Wesley, 2006. Digital image scanned by Google Books.
Original image by Bert Freudenberg, University of Magdeburg, 2002.

Particle systems on the GPU

Shaders extend the use of *texture memory* dramatically. Shaders can write to texture memory, and textures are no longer limited to being two-dimensional planes of RGB (A).

- A particle systems can be represented by storing a position and velocity for every particle.
- A fragment shader can render a particle system entirely in hardware by using texture memory to store and evolve particle data.

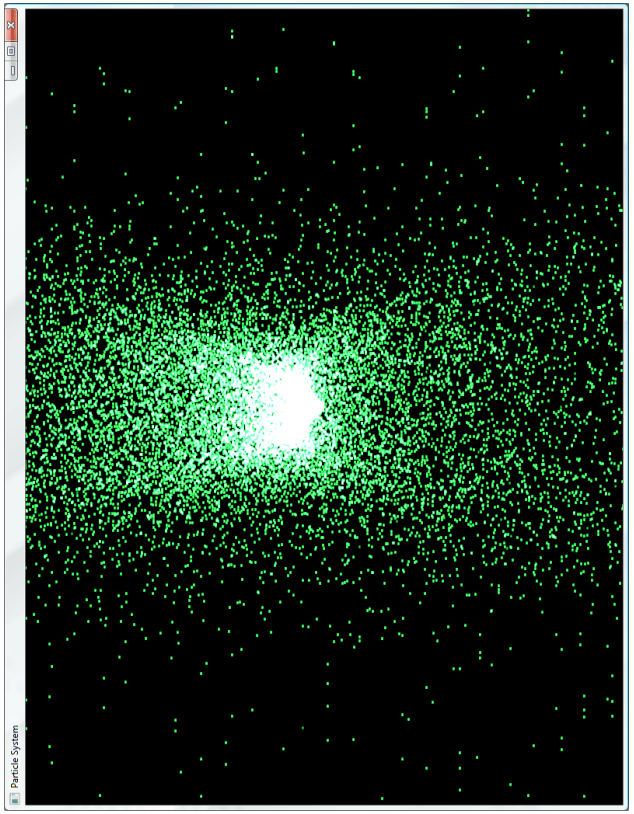


Image by Michael Short

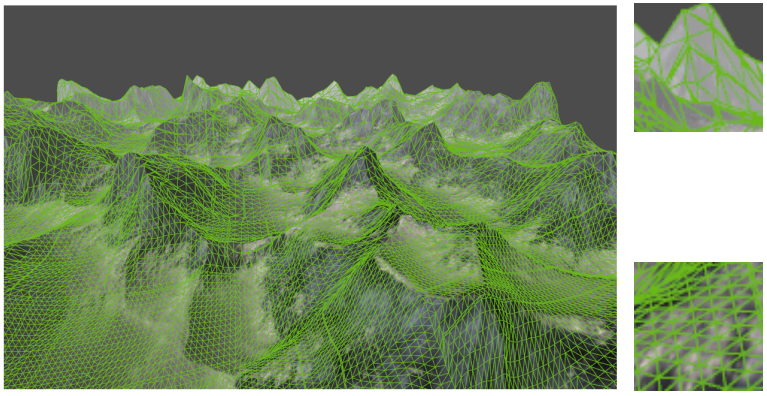
Tessellation shaders

Tesselation is a new shader type introduced in OpenGL 4.x. Tesselation shaders generate new vertices within *patches*, transforming a small number of vertices describing triangles or quads into a large number of vertices which can be positioned individually.

One use of tessellation is in rendering geometry such as game models or terrain with view-dependent *Levels of Detail* ("LOD").

Another is to do with geometry what ray-tracing did with bump-mapping: high-precision realtime geometric deformation.

Note how triangles are small and detailed close to the camera, but become very large and coarse in the distance.



Florian Boesch's LOD terrain demo
http://codeflow.org/entries/2010/nov/07/opengl-4-tessellation/

jabtunes.com's WebGL tessellation demo

Tessellation shaders

How it works:

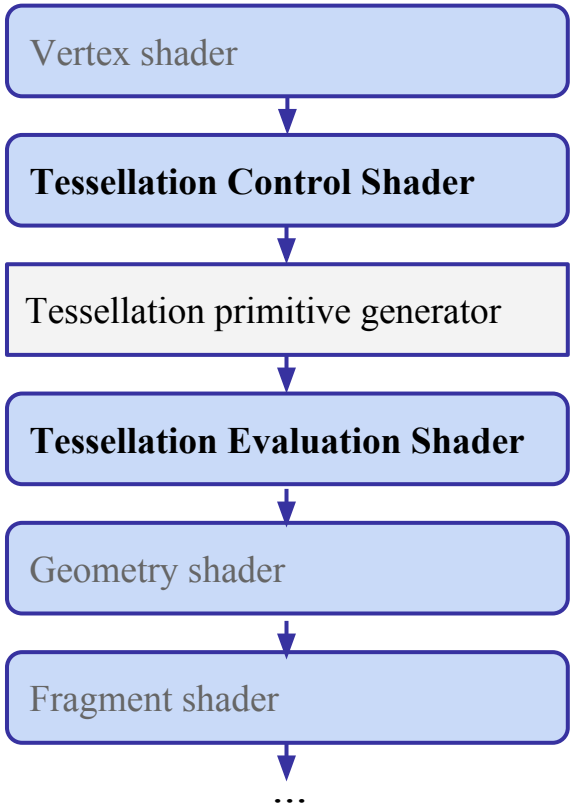
- You tell OpenGL how many vertices a single *patch* will have:

```
glPatchParameteri(GL_PATCH_VERTICES, 4);
```

- You tell OpenGL to render your patches:

```
glDrawArrays(GL_PATCHES, first, numVerts);
```

- The *Tessellation Control Shader* specifies output parameters defining how a patch is split up: `gl_TessLevelOuter[]` and `gl_TessLevelInner[]`.
  These control the number of vertices per primitive edge and the number of nested inner levels, respectively.

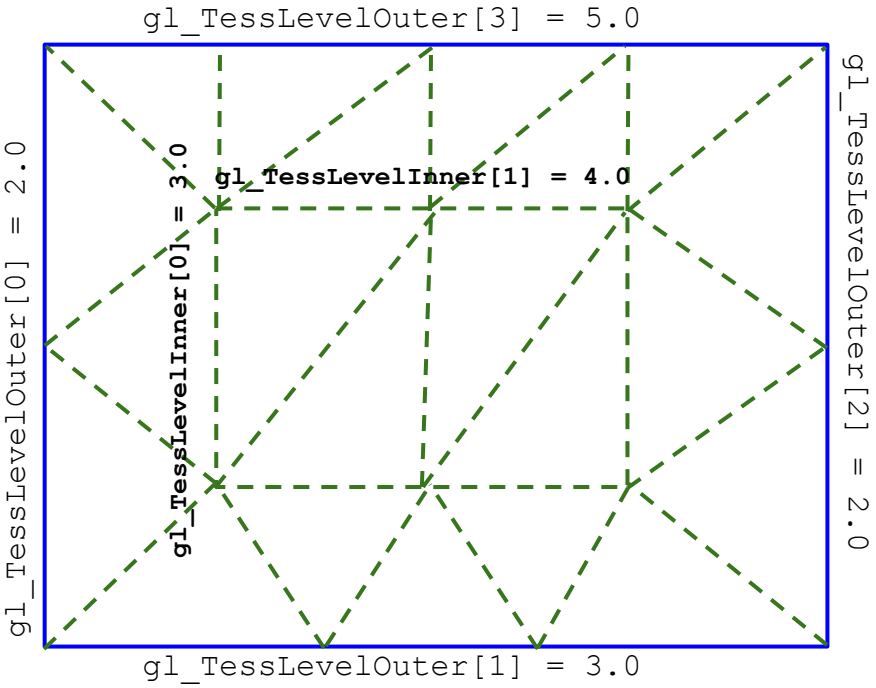


Tessellation shaders

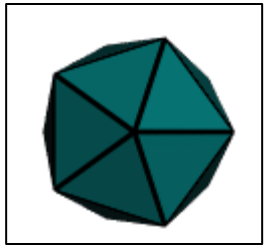
- The *tessellation primitive generator* generates new vertices along the outer edge and inside the patch, as specified by `gl_TessLevelOuter[]` and `gl_TessLevelInner[]`.

Each field is an array. Within the array, each value sets the number of intervals to generate during subprimitive generation.

Triangles are indexed similarly, but only use the first three `Outer` and the first `Inner` array field.



Tessellation shaders



- The generated vertices are then passed to the *Tesselation Evaluation Shader*, which can update vertex position, color, normal, and all other per-vertex data.
- Ultimately the complete set of new vertices is passed to the geometry and fragment shaders.

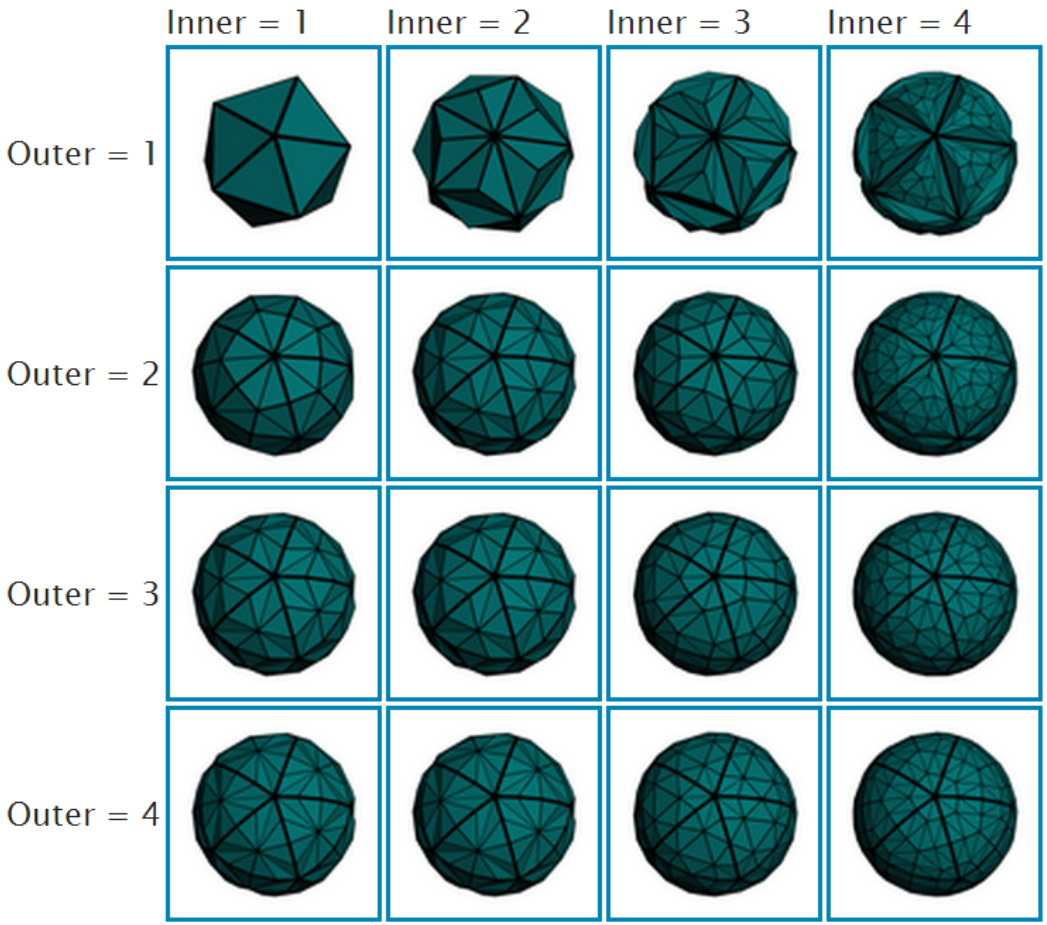
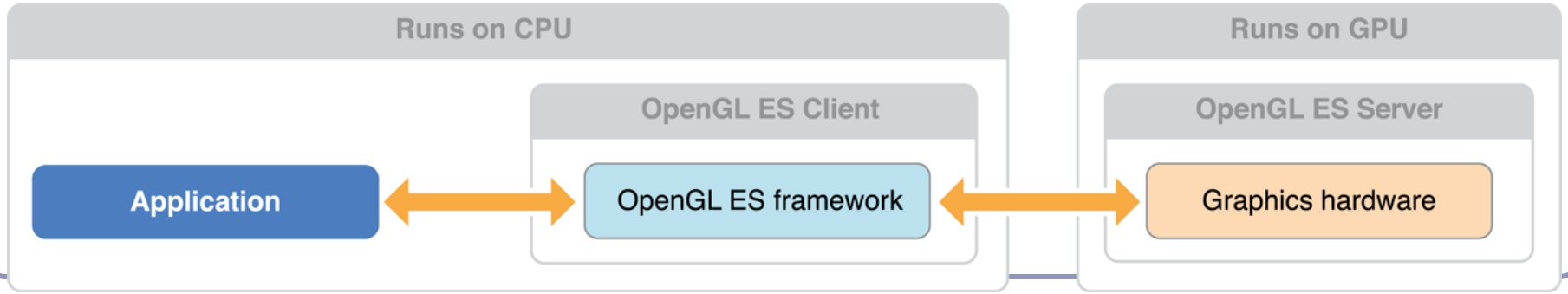


Image credit: Philip Rideout
http://prideout.net/blog/?p=48

OpenGL ES

OpenGL ES ("*Embedded Subsystem*") is a subset of OpenGL designed for constrained devices, like phones.

OpenGL ES 2.0 uses shaders to outsource work to the GPU, which enables very fast 3D. The OpenGL ES framework provides a *client/server* architecture which isolates the CPU from expensive graphics operations.



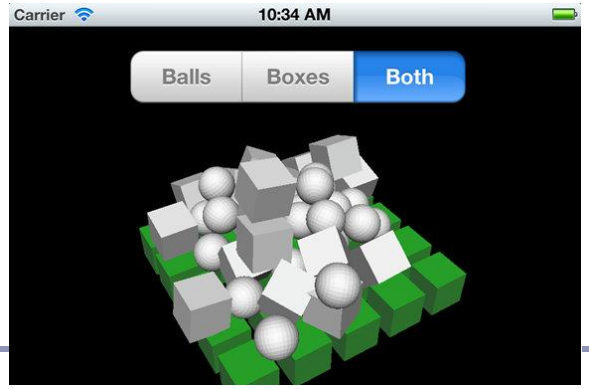
Architecture of an OpenGL ES application. From developer.apple.com

OpenGL ES

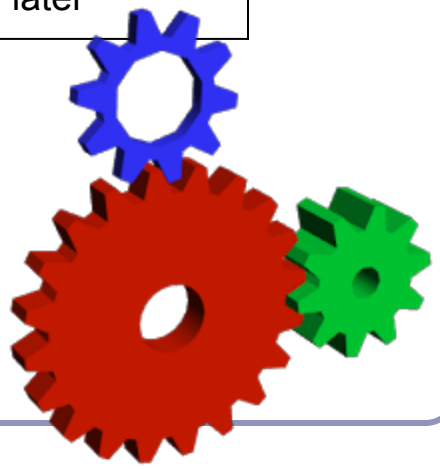
Widespread and evolving support

OpenGL ES	Android	iOS
OpenGL ES 1.0 and 1.1	Android 1.0 and higher	Apple iOS for iPad, iPhone, and iPod Touch
OpenGL ES 2.0	Android 2.2 (API level 8) and higher	Apple iOS 5 or later
OpenGL ES 3.0	Android 4.3 (API level 18)	Apple iOS 7 or later

Shadowgun (Android)



Physics Demo (iOS)



Gears Demo (Khronos Group)

Designed for Mobile

Key traits of OpenGL ES:

- Very small memory footprint
- Very low power consumption
- Smooth transitions from software rendering on low-end devices to hardware rendering on high-end; the developer should never have to worry
- Widespread industry adoption
- “Easy to use” and “well documented”, according to the authoring body

OpenGL ES is a subset of OpenGL

Includes

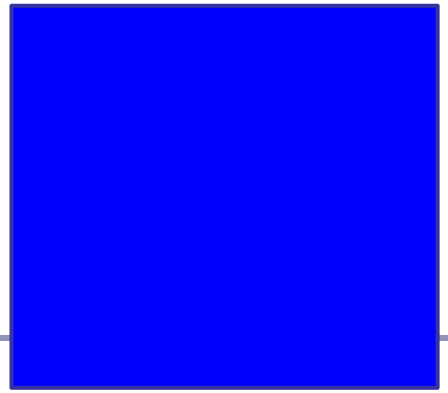
- Vertex shaders
- Fragment shaders
- Vertex buffers
- Textures
- Framebuffers
- Render states
- …

Does not include

- Geometry shaders
- Tessellation shaders
- Vertex Array Objects
- Multiple render targets
- Floating-point textures
- Compressed textures
- FS depth writes
- …

OpenGL ES - A minimal Android sample

```
public class OpenGLES20Activity extends Activity {

    private GLSurfaceView mGLView;

    @Override
    public void onCreate(Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);

        // Create a GLSurfaceView instance and set it
        // as the ContentView for this Activity.
        mGLView = new MySurfaceView(this);
        setContentView(mGLView);
    }
}
```



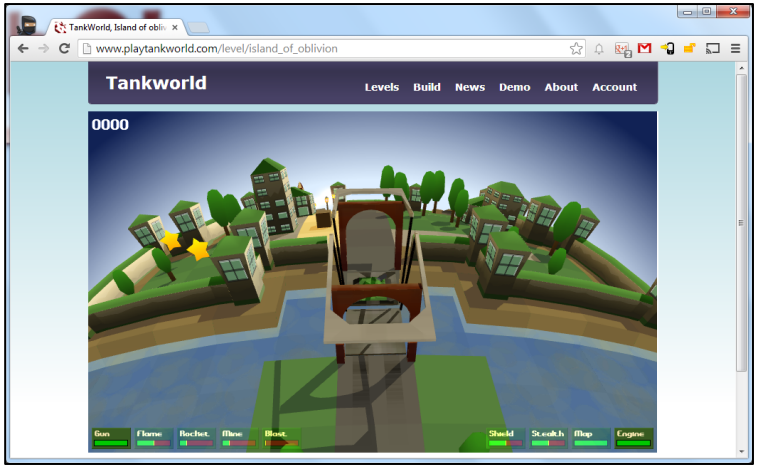
```
class MySurfaceView extends GLSurfaceView {
    public MyGLSurfaceView(Context context){
        super(context);
        // Set the Renderer for drawing on the GLSurfaceView
        setRenderer(new MyRenderer());
    }
}

public class MyRenderer implements GLSurfaceView.Renderer {
    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        // Set the background frame color
        GLES20.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    }
    public void onDrawFrame(GL10 unused) {
        // Redraw background color
        GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT);
    }
    public void onSurfaceChanged(GL10 gl, int width, int height) {
        GLES20.glViewport(0, 0, width, height);
    }
}
```

Source: Android.com, "Building an OpenGL ES Environment"
http://developer.android.com/training/graphics/opengl/environment.html

WebGL

WebGL is a port of OpenGL ES to the Web.

- It runs on Windows, Mac, and Linux
- It runs on desktop and mobile
- There's no plugins to install
- It runs on the GPU



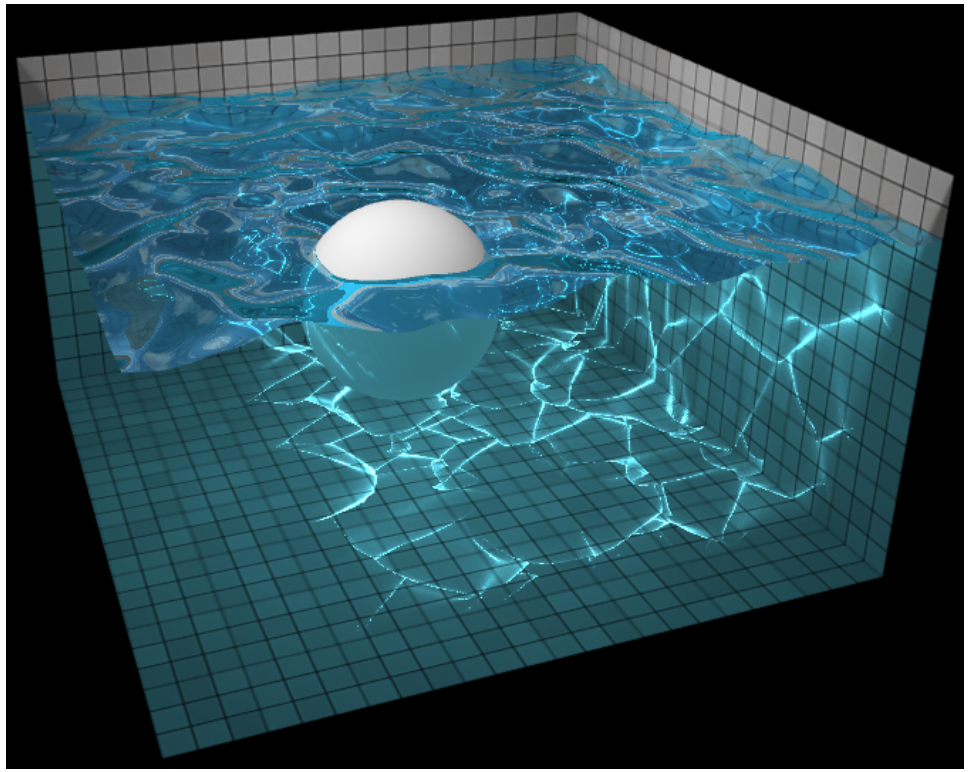
www.playtankworld.com

...and all it costs you is…

...you have to write all your code in JavaScript.

WebGL

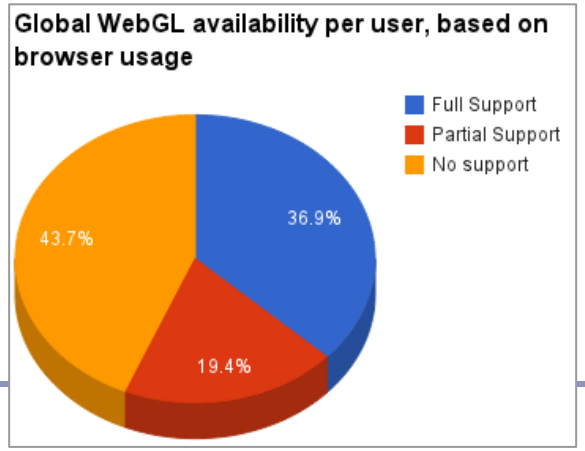
Demo: WebGL Water by Evan Wallace

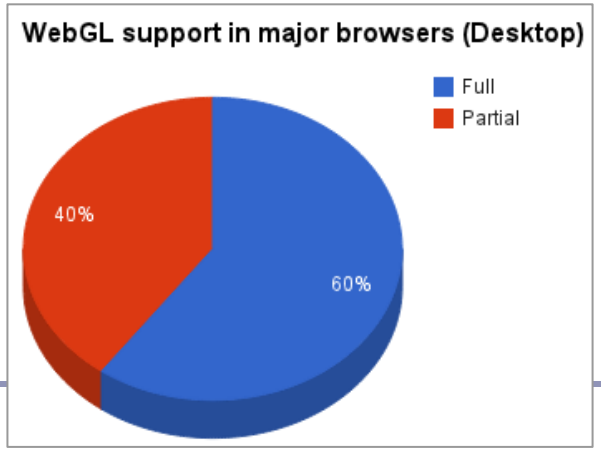


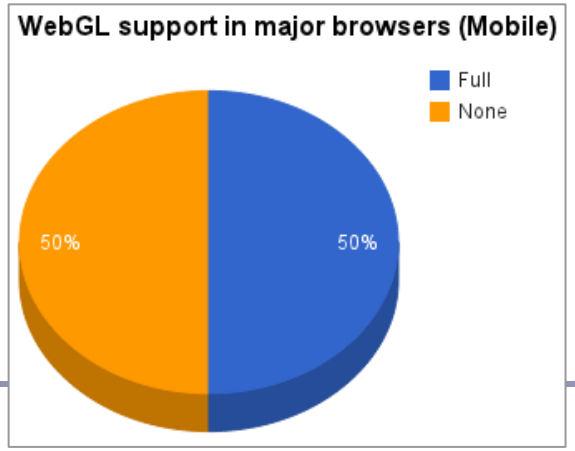
WebGL adoption

WebGL support by browser.
Data gathered Nov-Dec 2013
Source: caniuse.com/webgl

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	Opera Mobile	Chrome for Android	Firefox for Android	IE Mobile
4 versions back	7.0: None	22.0: Partial	27.0: Full	5.0: None	12.0: Partial	4.0-4.1: None		2.3: None		11.1: None			
3 versions back	8.0: None	23.0: Partial	28.0: Full	5.1: Partial	12.1: Partial	4.2-4.3: None		3.0: None		11.5: None			
2 versions back	9.0: None	24.0: Partial	29.0: Full	6.0: Partial	15.0: Full	5.0-5.1: None		4.0: None		12.0: Partial			
Previous version	10.0: None	25.0: Partial	30.0: Full	6.1: Partial	16.0: Full	6.0-6.1: None		4.1: None	7.0: None	12.1: Partial			
Current	11.0: Full	26.0: Partial	31.0: Full	7.0: Partial	17.0: Full	7.0: None	5.0-7.0: None	4.2-4.3: None	10.0: Full	16.0: Full	31.0: Full	25.0: Full	10.0: None

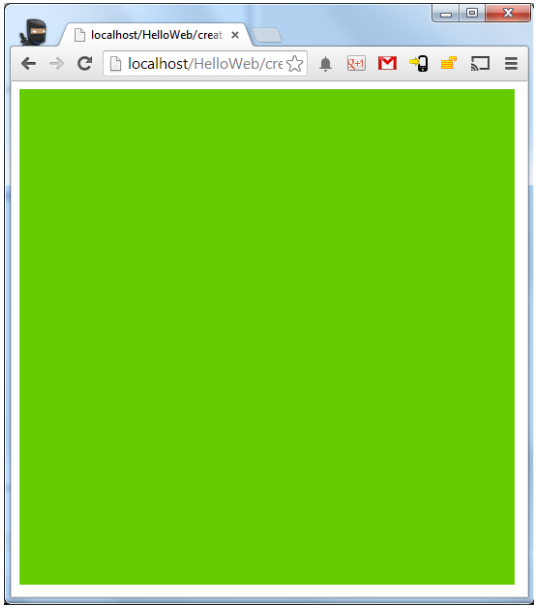






WebGL - Creating a Canvas

```
<html>
  <style type="text/css">
    canvas { background: blue; }
  </style>
  <script type="text/javascript"
      src="https://www.khronos.org/registry/webgl/sdk/demos/common/webgl-utils.js ">
  </script>
  <script type="text/javascript">
    window.onload = function() {
      var canvas = document.getElementById("gl-canvas");
      var gl = WebGLUtils.setupWebGL(canvas);
      if (!gl) { alert("WebGL isn't available"); }
      gl.viewport(0, 0, canvas.width, canvas.height);
      gl.clearColor(0.4, 0.8, 0.0, 1.0);
      gl.clear(gl.COLOR_BUFFER_BIT);
    }
  </script>
  <body>
    <canvas id="gl-canvas" width="512" height="512">
      Your browser doesn't support HTML5's
      Canvas element.
    </canvas>
  </body>
</html>
```



WebGL - Installing shaders

```
function getShader(gl, id) {
  var script = document.getElementById(id);
  var shader;
  if (script.type == "x-shader/x-vertex") {
    shader = gl.createShader(gl.VERTEX_SHADER);
  } else if (script.type == "x-shader/x-fragment")
{
    shader = gl.createShader(gl.FRAGMENT_SHADER);
  }
  gl.shaderSource(shader, script.text);
  gl.compileShader(shader);
  if (!gl.getShaderParameter(
      shader, gl.COMPILE_STATUS)) {
    alert(gl.getShaderInfoLog(shader));
    return null;
  }
  return shader;
}
```

To avoid cross-site security issues, shaders are often inlined into the HTML using the *x-shader* MIME types:

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

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

WebGL libraries abound

A number of libraries can ease the boilerplate:

- Three.js, used in many of the Chrome Experiments demos
- CopperLicht, a 3D world editor
- PhiloGL, focused on data visualization
- GLGE, used for early prototypes of Zygote Body
- SceneJS, highly detailed 3D visualisation

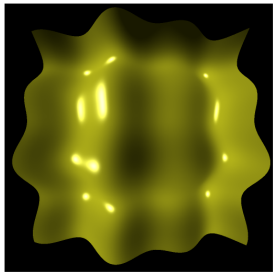
You can even pretend you're not using Javascript at all:

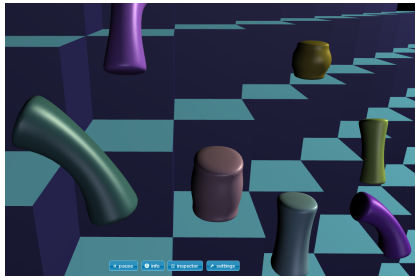
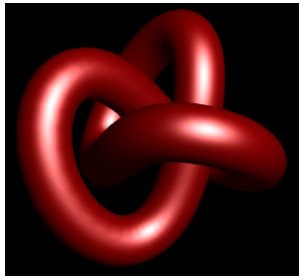
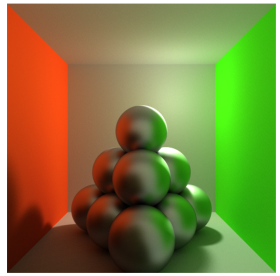
- GwtGL, a GWT (Java-to-Javascript) WebGL wrapper
- X3Dom, a set of 'HTML' tags for 3D and the spiritual descendant of VRML

WebGL Demos

WebGL demos abound:

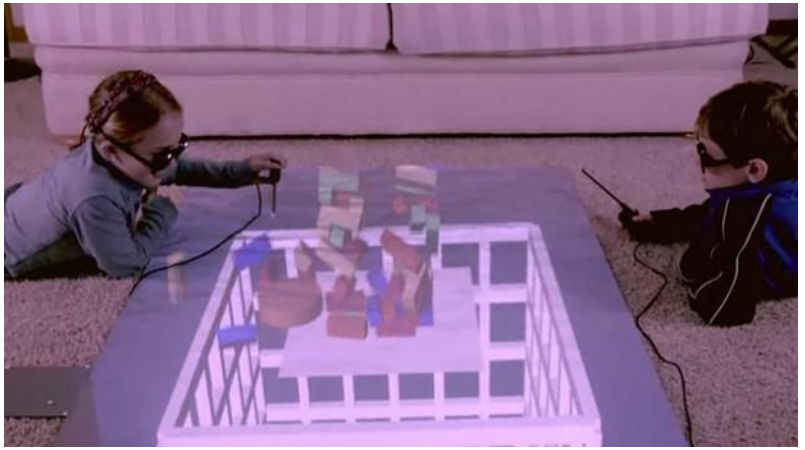
- NVIDIA's Vertex Buffer Object demo
- Jellyfish!
- "Sproingies"
- Dr. Thorsten Thormählen's Phong shaders demo
- Realtime ray-tracing on the GPU in WebGL



Augmented Reality (one example)

CastAR is an upcoming *augmented reality* system in which the wearer wears two tiny, lightweight projectors on their glasses. These projectors project onto a *retroreflective* material, which reflects light only to the wearer of the glasses, creating an illusory 3D volume within the canvas.



Virtual Reality: The Sword of Damocles (1968)

In 1968, Harvard Professor Ivan Sutherland, working with his student Bob Sproull, invented the world's first *head-mounted display*, or *HMD*. Their device combined three-dimensional binocular displays with head-tracking technology and was capable of placing the user in a virtual "room", a wire-frame enclosure which would re-orient and update as the user moved their head.

The head-mounted display weighed so much that the device had to be suspended from the ceiling with heavy scaffolding. The way it hung perilously above the user gave rise to its nickname, "The Sword of Damocles".

Virtual Reality: The Oculus Rift (2014)

The *Oculus Rift* is a head-mounted display being produced by OculusVR, Inc. It should be released in 2014.

The Rift combines a lightweight display, carefully-positioned lenses, and a sensitive head-tracker to produce an immersive virtual reality experience.

How the Oculus Rift works

The Rift locks two adjustable lenses in front of an LCD display (1280x800, with an HD version planned.)

The position and orientation of the lenses can be queried from software.

Images: http://www.pcauthority.com.au

How the Oculus Rift works

Applications targeting the Rift as a display generate a single 1280x800 image which is the built from two 640x800 images. Rendering the two side-by-side on the same screen leads the wearer's brain to composite them into a single image.

Varying the camera position and angle used to generate the two images delivers a 3D effect.

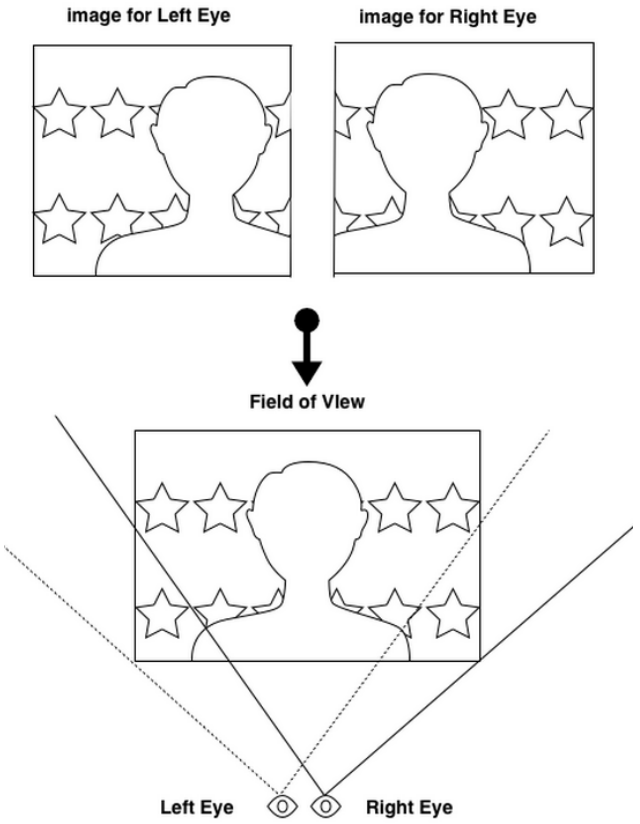


Image: Bryla and Davis, *Oculus Rift in Action*, Manning Press, 2014

How the Oculus Rift works

Lenses bend light: the lenses in the Rift warp the image on the screen, creating a *pincushion distortion*.

This is countered by introducing a *barrel distortion* in the shader used to render the image.

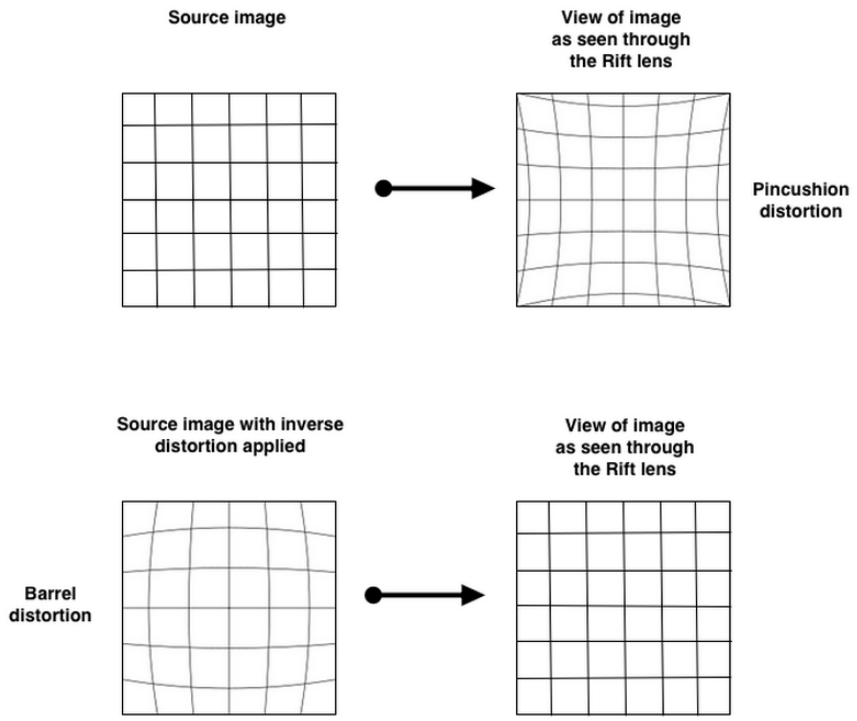
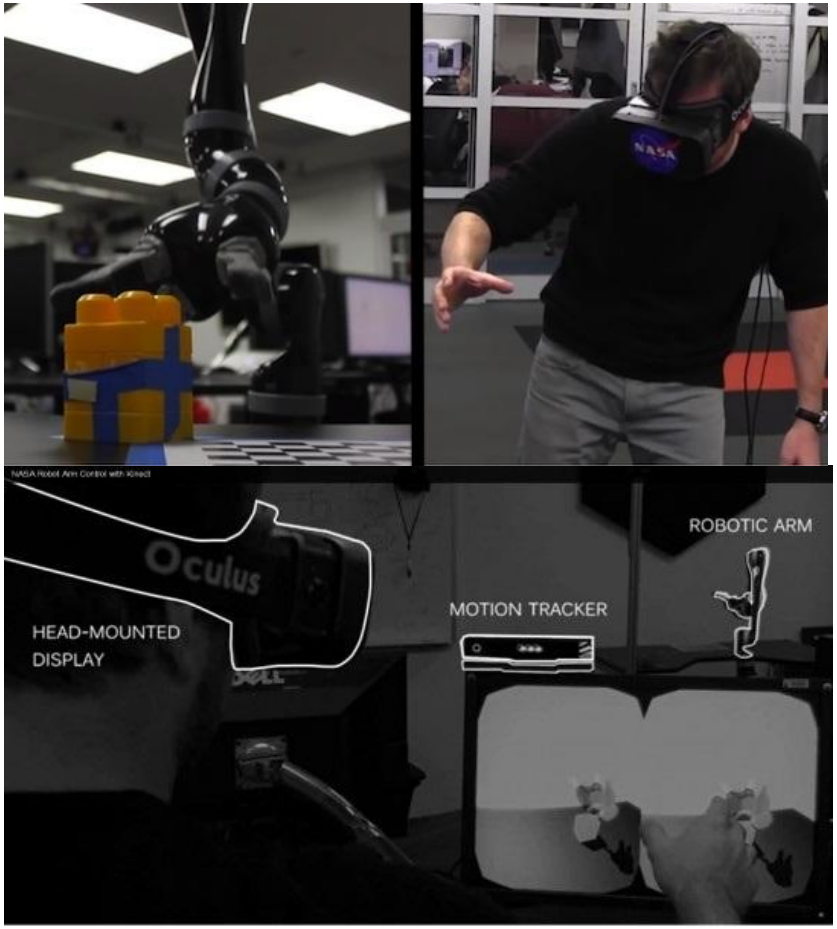


Image: Bryla and Davis, *Oculus Rift in Action*, Manning Press, 2014

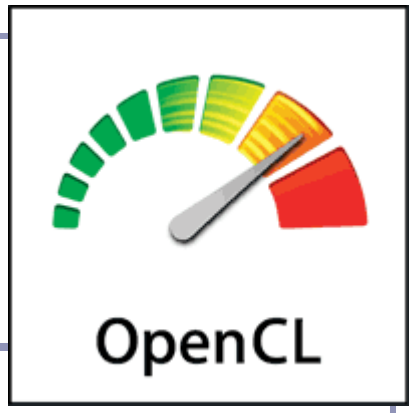
Putting the Rift to work

NASA has built an integrated telepresence experience using a Kinnect 2 and an Oculus Rift.

They hope that despite high latency, by using gestures to indicate intent, earth-bound users could remotely control robots in real time in space.



OpenCL



OpenCL, the Open Computing Language, exposes the computing power of the GPU to non-graphical programs.

From the Khronos Group site:

"OpenCL™ is the first open, royalty-free standard for cross-platform, parallel programming of modern processors found in personal computers, servers and handheld/embedded devices."

"OpenCL 2.0 defines an enhanced execution model and a subset of the C11 and C++11 memory model, synchronization and atomic operations."

Versions even work with OpenGL ES and WebGL: WebCL enables high-powered computing in JavaScript.

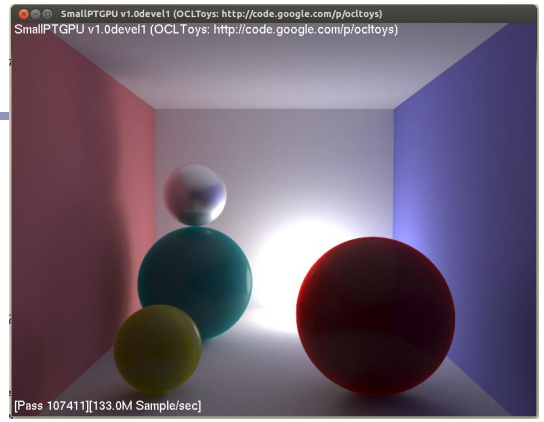
OpenCL

Designed to:

- Support parallel execution on single or multiple processors
  - GPU, CPU, GPU + CPU or multiple GPUs
- Offer Desktop and Handheld Profiles
- Offer shared virtual memory space
- Work with graphics APIs such as OpenGL
- Accelerate those same APIs--OpenGL uses OpenCL now

To work with OpenCL you'll need to download and install drivers specific to your OS, CPU and GPU.

- Cross-platform portability is an ongoing project goal, dependent on adoption.



OpenCL ray tracer
https://code.google.com/p/ocltoys/

Realtime particle fluids - FSU
Ian Johnson, Gordon Erlebacher
http://enja.org/2011/03/31/particles-in-bge-improved-code-collisions-and-hose/

CPU vs GPU – an object demonstration

"NVIDIA: Adam and Jamie explain parallel processing on the GPU"
http://www.youtube.com/watch?v=ZrJeYFxpUyQ

References

WebGl:

http://khronosgroup.github.io/siggraph2012course/CanvasCSSAndWebGL/webgl.html

https://code.google.com/p/gwtgl/

http://www.mathematik.uni-marburg.de/~thormae/lectures/graphics1/code/WebGLShaderLightMat/ShaderLightMat.html

http://www.zygotebody.com

Particle systems:

http://www.gamasutra.com/view/feature/130535/building_a_millionparticle_system.php?print=1

Tessellation:

http://prideout.net/blog/?p=48

http://antongerdelan.net/opengl/tessellation.html

OpenCL:

http://s08.idav.ucdavis.edu/munshi-opencl.pdf

https://www.khronos.org/opencl/