Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

Framebuffer Objects

Ed Angel
Professor Emeritus of Computer Science
University of New Mexico

Objectives

- Look at methods that use memory on the graphics card
- Introduce off screen rendering
- Learn how to create framebuffer objects
  - Create a renderbuffer
  - Attach resources

Discrete Processing in WebGL

- Recent GPUs contain large amounts of memory
  - Texture memory
  - Framebuffer
  - Floating point
- Fragment shaders support discrete operations at the pixel level
- Separate pixel (texel) pipeline

Accessing the Framebuffer

- Pre 3.1 OpenGL had functions that allowed access to the framebuffer and other OpenGL buffers
  - Draw Pixels
  - Read Pixels
  - Copy Pixels
  - BitBlt
  - Accumulation Buffer functions
- All deprecated

Going between CPU and GPU

- We have already seen that we can write pixels as texels to texture memory
- Texture objects reduce transfers between CPU and GPU
- Transfer of pixel data back to CPU slow
- Want to manipulate pixels without going back to CPU
  - Image processing
  - GPGPU

Framebuffer Objects

- Framebuffer Objects (FBOs) are buffers that are created by the application
  - Not under control of window system
  - Cannot be displayed
  - Can attach a renderbuffer to a FBO and can render off screen into the attached buffer
  - Attached buffer can then be detached and used as a texture map for an on-screen render to the default frame buffer

Render to Texture

- Textures are shared by all instances of the fragment shader
- If we render to a texture attachment we can create a new texture image that can be used in subsequent renderings
- Use a double buffering strategy for operations such as convolution

Steps

- Create an Empty Texture Object
- Create a FBO
- Attach renderbuffer for texture image
- Bind FBO
- Render scene
- Detach renderbuffer
- Bind texture
- Render with new texture

Empty Texture Object

```
texture1 = gl.createTexture();
gl.activeTexture( gl.TEXTURE0 );
gl.bindTexture( gl.TEXTURE_2D, texture1 );

gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, 512, 512, 0, gl.RGBA,
      gl.UNSIGNED_BYTE, null);

gl.generateMipmap(gl.TEXTURE_2D);    gl.texParameteri(
gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER,
      gl.NEAREST_MIPMAP_LINEAR );    gl.texParameteri(
gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER,
      gl.NEAREST )
```

Creating a FBO

- We create a framebuffer object in a similar manner to other objects
- Creating an FBO creates an empty FBO
- Must add needed resources
  - Can add a renderbuffer to render into
  - Can add a texture which can also be rendered into
  - For hidden surface removal we must add a depth buffer attachment to the renderbuffer

Frame Buffer Object

```
var framebuffer = gl.createFramebuffer();
gl.bindFramebuffer(gl.FRAMEBUFFER, framebuffer);
framebuffer.width = 512;
framebuffer.height = 512;
//renderbuffer = gl.createRenderbuffer();
//gl.bindRenderbuffer(gl.RENDERBUFFER, renderbuffer);
//gl.renderbufferStorage(gl.RENDERBUFFER,
//      gl.DEPTH_COMPONENT16, 512, 512);
// Attach color buffer
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
        gl.TEXTURE_2D, texture1, 0);
//gl.framebufferRenderbuffer(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT,
        gl.RENDERBUFFER, renderbuffer);
// check for completeness
  var status = gl.checkFramebufferStatus(gl.FRAMEBUFFER);
  if(status != gl.FRAMEBUFFER_COMPLETE) alert('Frame Buffer Not Complete');
```

Rest of Initialization

- Same as previous examples
  - Allocate VAO (Vertex-Array Object)
  - Fill VAO with data for render to texture
- Initialize two program objects with different shaders
  - First for render to texture
  - Second for rendering with created texture

Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

Render to Texture

Ed Angel
Professor Emeritus of Computer Science
University of New Mexico

Objectives

- Examples of render-to-texture
- Render a triangle to texture, then use this texture on a rectangle
- Introduce buffer pingponging

Program Objects and Shaders

- For most applications of render-to-texture we need multiple program objects and shaders
  - One set for creating a texture
  - Second set for rendering with that texture
- Applications that we consider later such as buffer pingponging may require additional program objects

Program Object 1 Shaders

pass through vertex shader:

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

fragment shader to get a red triangle:

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

Program Object 2 Shaders

```
// vertex shader

attribute vec4 vPosition;
attribute vec2 vTexCoord;
varying vec2 fTexCoord;
void main()
{
gl_Position = vPosition;
fTexCoord = vTexCoord;
}
```

```
// fragment shader

precision mediump float;

varying  vec2 fTexCoord;
uniform sampler2D texture;
void main()
{
    gl_FragColor = texture2D( texture,
                fTexCoord);
}
```

First Render (to Texture)

```
gl.useProgram( program1 );
var buffer1 = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, buffer1 );
gl.bufferData( gl.ARRAY_BUFFER, flatten(pointsArray), gl.STATIC_DRAW );

// Initialize the vertex position attribute from the vertex shader

var vPosition = gl.getAttribLocation( program1, "vPosition" );
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );

// Render one triangle
gl.viewport(0, 0, 64, 64);
gl.clearColor(0.5, 0.5, 0.5, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT );
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Set Up Second Render

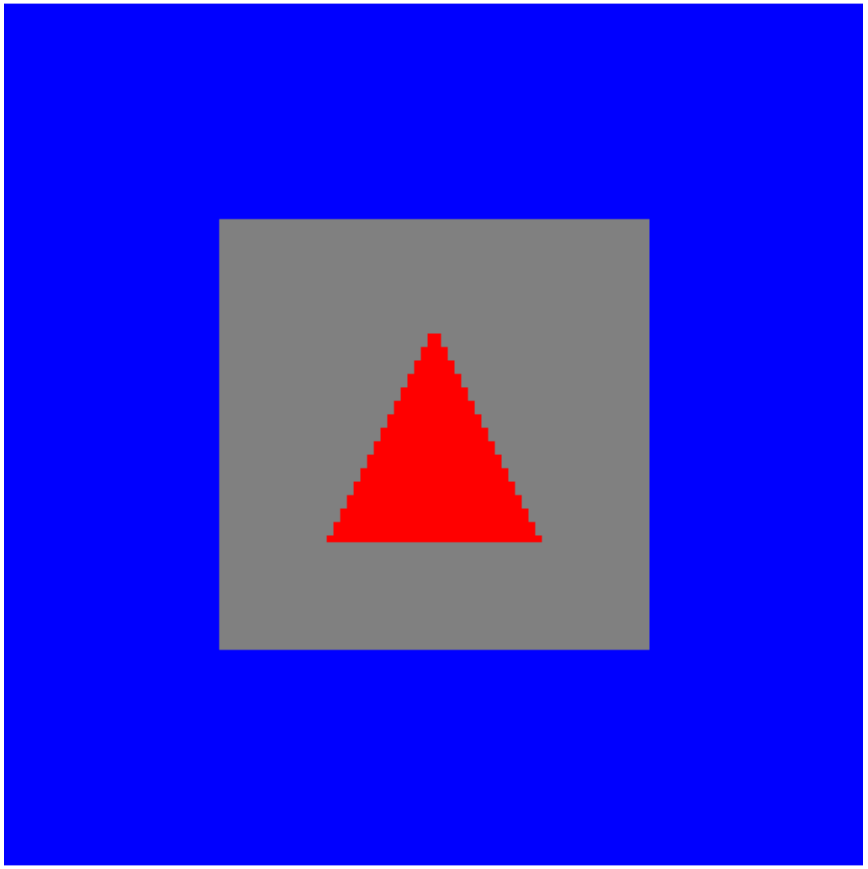
```
// Bind to default window system framebuffer

    gl.bindFramebuffer(gl.FRAMEBUFFER, null);
    gl.disableVertexAttribArray(vPosition);
    gl.useProgram(program2);

// Assume we have already set up a texture object with null texture image

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, texture1);

// set up vertex attribute arrays for texture coordinates and rectangle as usual
```

Data for Second Render

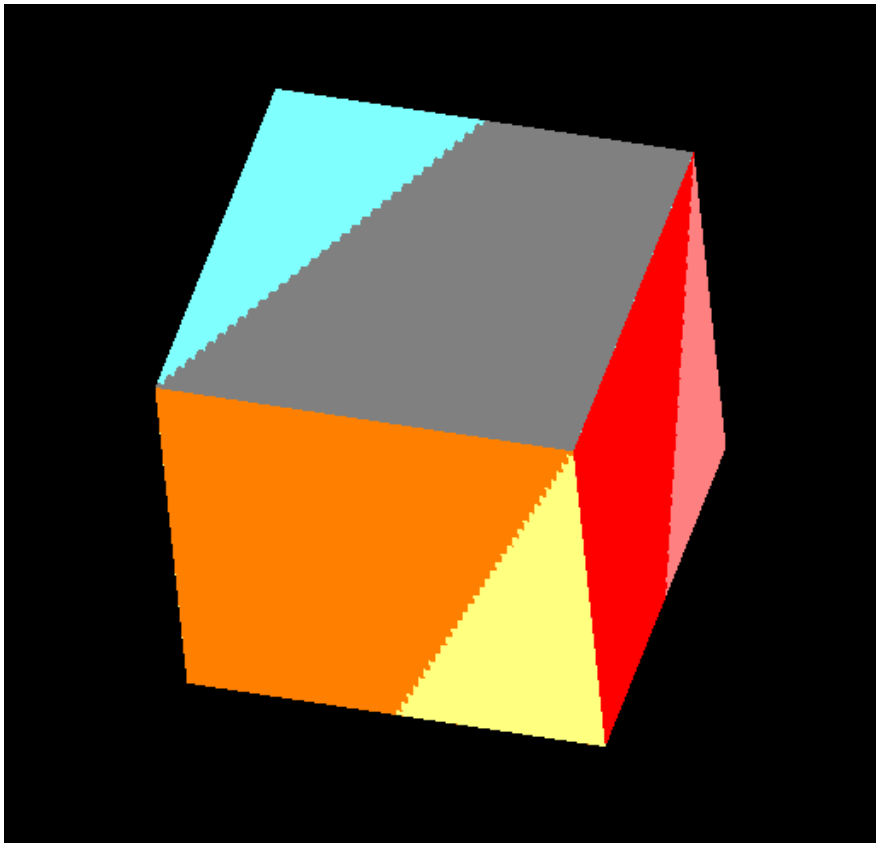
```
var buffer2 = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, buffer2);
gl.bufferData(gl.ARRAY_BUFFER,   new flatten(vertices),
            gl.STATIC_DRAW);

var vPosition = gl.getAttribLocation( program2, "vPosition" );
gl.vertexAttribPointer( vPosition, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vPosition );

var buffer3 = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, buffer3);
gl.bufferData( gl.ARRAY_BUFFER,   flatten(texCoord), gl.STATIC_DRAW);

var vTexCoord = gl.getAttribLocation( program2, "vTexCoord");
gl.vertexAttribPointer( vTexCoord, 2, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vTexCoord );
```

Render a Quad with Texture

```
gl.uniform1i( gl.getUniformLocation(program2, "texture"), 0);

gl.viewport(0, 0, 512, 512);
gl.clearColor(0.0, 0.0, 1.0, 1.0);
gl.clear( gl.COLOR_BUFFER_BIT );

gl.drawArrays(gl.TRIANGLES, 0, 6);
```



Dynamic 3D Example



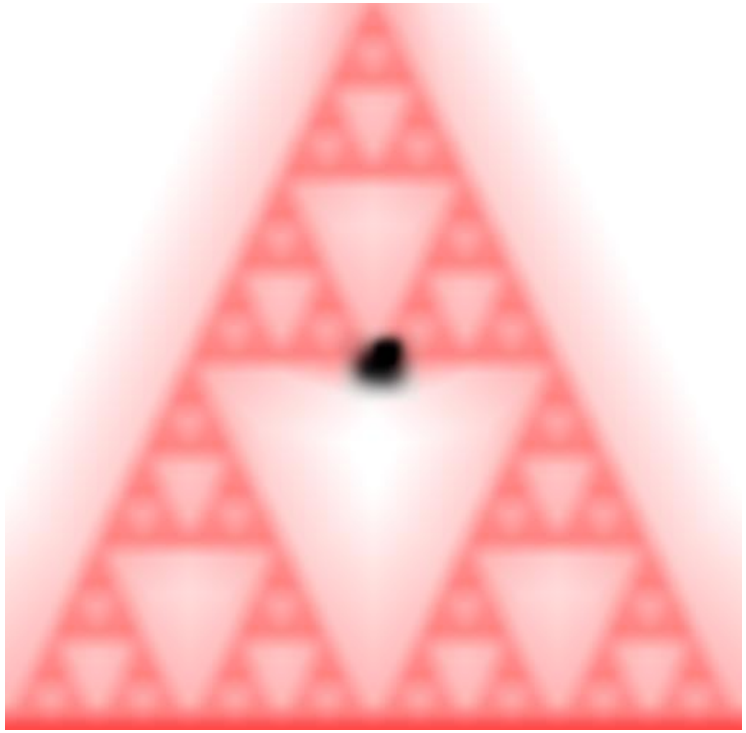
Buffer Ping-ponging

- Iterative calculations can be accomplished using multiple render buffers
- Original data in texture buffer 1
- Render to texture buffer 2
- Swap buffers and rerender to texture

Particle System Example

- Random motion of a particle
  - Render as a point
  - Diffuse rendered image to create motion blur effect
  - Insert particle again in new position
- Example use Sierpinski gasket as initial background
- Uses three program objects

Screen Shots



Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

Agent Based Models

Ed Angel

Professor Emeritus of Computer Science

University of New Mexico

Objectives

- Introduce a powerful form of simulation
- Use render-to-texture for dynamic simulations using agent-based models
- Example of diffusion

Agent Based Models (ABMs)

- Consider a particle system in which particle can be programmed with individual behaviors and properties
  - different colors
  - different geometry
  - different rules
- Agents can interact with each other and with the environment

Simulating Ant Behavior

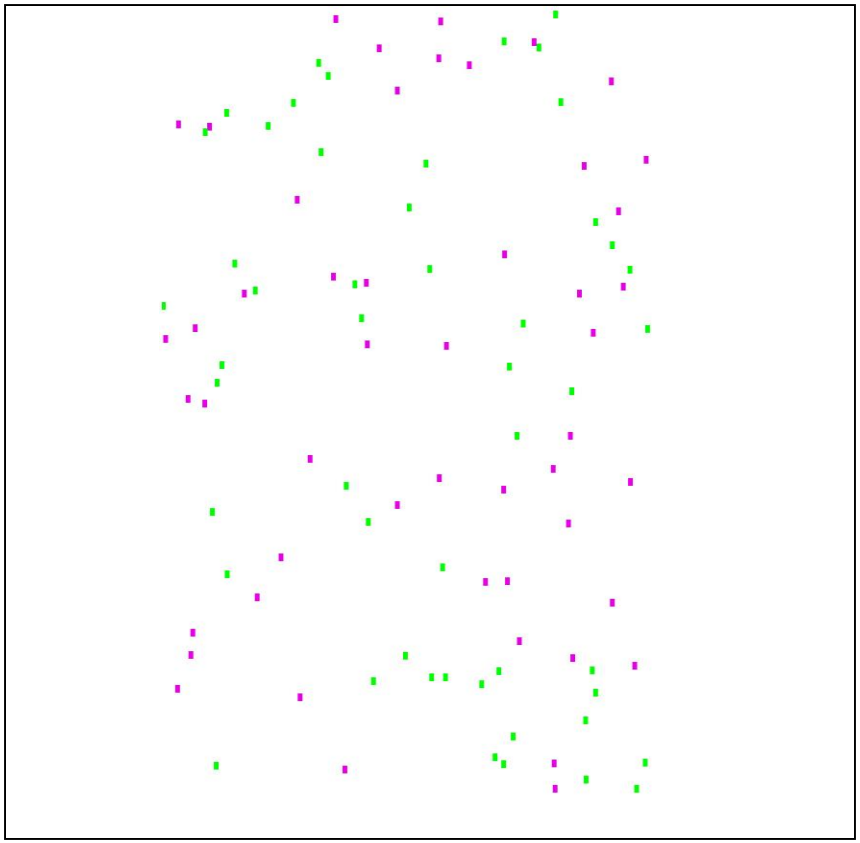
- Consider ants searching for food
- At the beginning, an ant moves randomly around the terrain searching for food
  - The ant can leave a chemical marker called a pheromone to indicate the spot was visited
  - Once food is found, other ants can trace the path by following the pheromone trail
- Model each ant as a point moving over a surface
- Render each point with arbitrary geometry

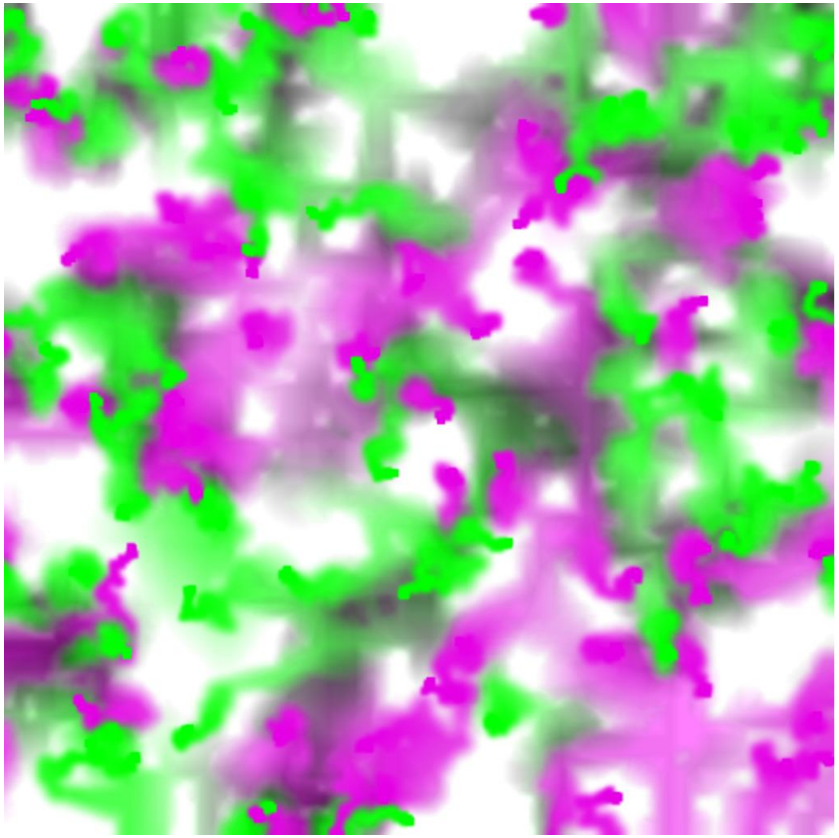
Diffusion Example I

- Two types of agents
  - no interaction with environment
  - differ only in color
- All move randomly
- Leave position information
  - need render-to-texture
- Diffuse position information
  - need buffer pingponging

The University of New Mexico

Snapshots





Initialization

- We need two program objects
  - One for rendering points in new positions
  - One for diffusing texture map
- Initialization is standard otherwise
  - setup texture objects
  - setup framebuffer object
  - distribute particles in random locations

Vertex Shader 1

```
attribute vec4 vPosition1;
attribute vec2 vTexCoord;
varying vec2 fTexCoord;
void main()
{
    gl_Position = vPosition1;
    fTexCoord = vTexCoord;
}
```

Fragment Shader 1

```
precision mediump float;
uniform sampler2D texture;
uniform float d;
uniform float s;
varying vec2 fTexCoord;
void main()
 {
   float x = fTexCoord.x;
   float y = fTexCoord.y;
   gl_FragColor = (texture2D( texture, vec2(x+d, y))
                +texture2D( texture, vec2(x, y+d))
                +texture2D( texture, vec2(x-d, y))
                +texture2D( texture, vec2(x, y-d)))/s;
 }
```

Vertex Shader 2

```
attribute  vec4 vPosition2;
uniform float pointSize;
void main()
 {
    gl_PointSize = pointSize;
    gl_Position = vPosition2;
}
```

Fragment Shader 2

```
precision mediump float;
uniform vec4 color;
void main()
{
    gl_FragColor = color;
}
```

Rendering Loop I

```
var render = function(){
   // render to texture
   // first a rectangle that is texture mapped
    gl.useProgram(program1);
    gl.bindFramebuffer( gl.FRAMEBUFFER, framebuffer);
    if(flag) {
        gl.bindTexture(gl.TEXTURE_2D, texture1);
        gl.framebufferTexture2D(gl.FRAMEBUFFER,
        gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, texture2, 0);
    }
    else {
        gl.bindTexture(gl.TEXTURE_2D, texture2);
        gl.framebufferTexture2D(gl.FRAMEBUFFER,
           gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, texture1, 0);
    }
   gl.drawArrays( gl.TRIANGLE_STRIP, 0, 4 );
```

Rendering Loop II

```
// render points
    gl.useProgram(program2);
    gl.vertexAttribPointer( vPosition2, 2, gl.FLOAT, false, 0, 0);
    gl.uniform4f( gl.getUniformLocation(program2, "color"), 0.9, 0.0, 0.9, 1.0);
     gl.drawArrays(gl.POINTS, 4, numPoints/2);
    gl.uniform4f( gl.getUniformLocation(program2, "color"), 0.0, 9.0, 0.0, 1.0);
    gl.drawArrays(gl.POINTS, 4+numPoints/2, numPoints/2);
 // render to display
    gl.useProgram(program1);
    gl.vertexAttribPointer( texLoc, 2, gl.FLOAT, false, 0, 32+8*numPoints);
    gl.generateMipmap(gl.TEXTURE_2D);
    gl.bindFramebuffer(gl.FRAMEBUFFER, null);
// pick texture
    if(flag) gl.bindTexture(gl.TEXTURE_2D, texture2);
    else gl.bindTexture(gl.TEXTURE_2D, texture1);
```

Rendering Loop III

```
    var r = 1024/texSize;
    gl.viewport(0, 0, r*texSize, r*texSize);
    gl.clear( gl.COLOR_BUFFER_BIT );
    gl.drawArrays( gl.TRIANGLE_STRIP, 0, 4 );
    gl.viewport(0, 0, texSize, texSize);
    gl.useProgram(program2);
// move particles in a random direction with wrap around
    for(var i=0; i<numPoints; i++) {
        vertices[4+i][0] += 0.01*(2.0*Math.random()-1.0);
       vertices[4+i][1] += 0.01*(2.0*Math.random()-1.0);
       if(vertices[4+i][0]>1.0) vertices[4+i][0]-= 2.0;
       if(vertices[4+i][0]<-1.0) vertices[4+i][0]+= 2.0;
       if(vertices[4+i][1]>1.0) vertices[4+i][1]-= 2.0;
       if(vertices[4+i][1]<-1.0) vertices[4+i][1]+= 2.0;
}
gl.bufferSubData(gl.ARRAY_BUFFER,  0, flatten(vertices));
```

Rendering Loop IV

```
// swap textures
    flag = !flag;
    requestAnimFrame(render);
}
```

Add Agent Behavior

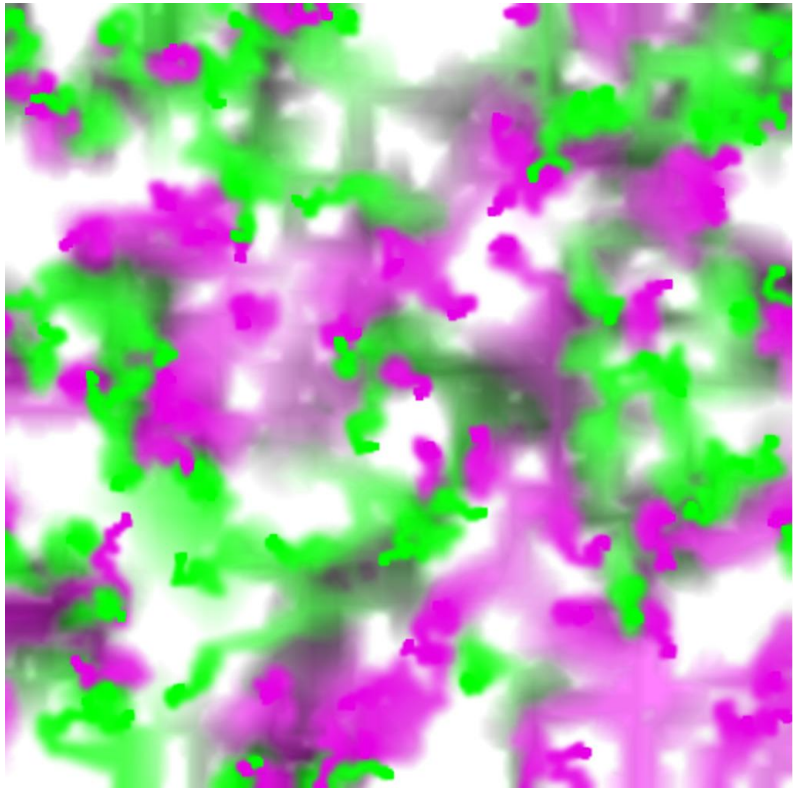
- Move randomly
- Check color where particle is located
- If green particle sees a green component over 128 move to (0.5, 0.5)
- If magenta particle sees a red component over 128 move to (-0.5, -0.5)

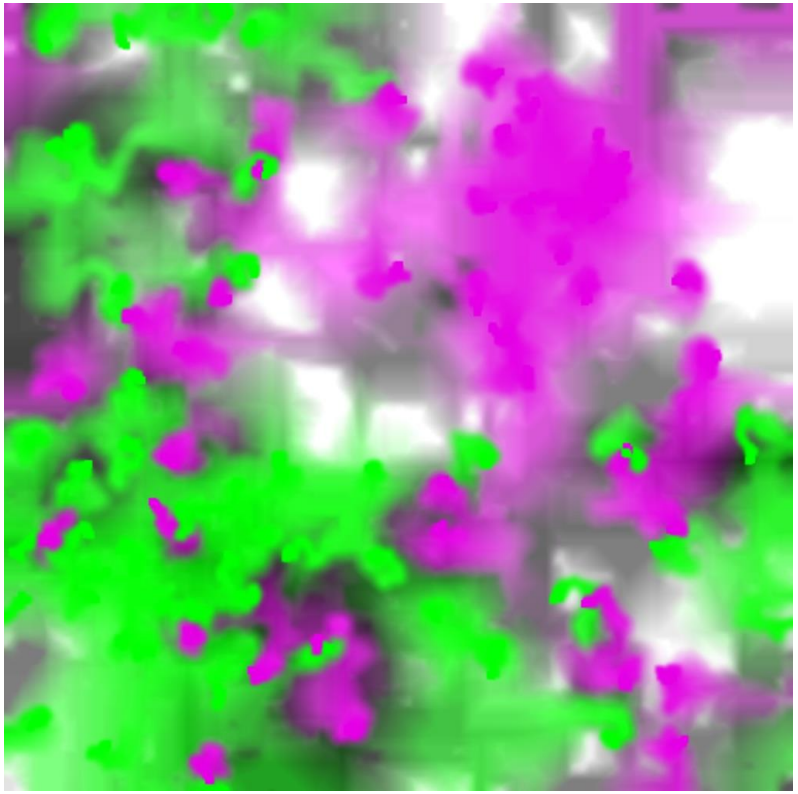
Diffusion Code

```
var color = new Uint8(4);
for(var i=0; i<numPoints/2; i++) {
        var x = Math.floor(511*(vertices[4+i][0]));
        var y = Math.floor(511*(vertices[4+i][1]));
        gl.readPixels(x, y, 1, 1, gl.RGBA, gl.UNSIGNED_BYTE, color);
       if(color[0]>128) {
             vertices[4+i][0] = 0.5;
             vertices[4+i][1] = 0.5;
    }
    for(var i=numPoints/2; i<numPoints; i++) {
        var x = Math.floor(511*(vertices[4+i][0]));
        var y = Math.floor(511*(vertices[4+i][1]));
        gl.readPixels(x, y, 1, 1, gl.RGBA, gl.UNSIGNED_BYTE, color);
        if(color[1]>128) {
            vertices[4+i][0] = -0.5;
            vertices[4+i][1] = -0.5;
    }
```

The University of New Mexico

Snapshots





without reading color

with reading color

Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

Picking by Color

Ed Angel
Professor Emeritus of Computer Science
University of New Mexico

Objectives

- Use off-screen rendering for picking
- Example: rotating cube with shading
  - indicate which face is clicked on with mouse
  - normal rendering uses vertex colors that are interpolated across each face
  - Vertex colors could be determined by lighting calculation or just assigned
  - use console log to indicate which face (or background) was clicked

Algorithm

- Assign a unique color to each object
- When the mouse is clicked:
  - Do an off-screen render using these colors and no lighting
  - use gl.readPixels to obtain the color of the pixel where the mouse is located
  - map the color to the object id
  - do a normal render to the display

Shaders

- Only need one program object
- Vertex shader: same as in previous cube examples
  - includes rotation matrices
  - gets angle as uniform variable
- Fragment shader
  - Stores face colors for picking
  - Gets vertex color for normal render from rasterizer
- Send uniform integer to fragment shader as index for desired color

Fragment Shader

```
precision mediump float;

uniform int i;
varying vec4 fColor;
void main()
{   vec4 c[7];
    c[0] = fColor;
    c[1] = vec4(1.0, 0.0, 0.0, 1.0);
    c[2] = vec4(0.0, 1.0, 0.0, 1.0);
    c[3] = vec4(0.0, 0.0, 1.0, 1.0);
    c[4] = vec4(1.0, 1.0, 0.0, 1.0);
    c[5] = vec4(0.0, 1.0, 1.0, 1.0);
    c[6] = vec4(1.0, 0.0, 1.0, 1.0);
```

Fragment Shader

```
// no case statement in GLSL

   if(i==0) gl_FragColor = c[0];
   else if(i==1) gl_FragColor = c[1];
   else if(i==2) gl_FragColor = c[2];
   else if(i==3) gl_FragColor = c[3];
   else if(i==4) gl_FragColor = c[4];
   else if(i==5) gl_FragColor = c[5];
   else if(i==6) gl_FragColor = c[6];
}
```

Setup

```
// Allocate a frame buffer object
   framebuffer = gl.createFramebuffer();
   gl.bindFramebuffer( gl.FRAMEBUFFER, framebuffer);
// Attach color buffer
   gl.framebufferTexture2D(gl.FRAMEBUFFER,
 gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, texture, 0);
 gl.bindFramebuffer(gl.FRAMEBUFFER, null);
```

Event Listener

```
canvas.addEventListener("mousedown", function(){
        gl.bindFramebuffer(gl.FRAMEBUFFER, framebuffer);
    gl.clear( gl.COLOR_BUFFER_BIT);
    gl.uniform3fv(thetaLoc, theta);
    for(var i=0; i<6; i++) {
       gl.uniform1i(gl.getUniformLocation(program, "i"), i+1);
       gl.drawArrays( gl.TRIANGLES, 6*i, 6 );
    }
    var x = event.clientX;
    var y = canvas.height -event.clientY;
    gl.readPixels(x, y, 1, 1, gl.RGBA,
          gl.UNSIGNED_BYTE, color);
```

Event Listener

```
if(color[0]==255)
        if(color[1]==255) console.log("yellow");
        else if(color[2]==255) console.log("magenta");
        else console.log("red");

else if(color[1]==255)
        if(color[2]==255) console.log("cyan");
        else console.log("green");

else if(color[2]==255) console.log("blue");
        else console.log("background");
```

Event Listener

```
 // return to default framebuffer
        gl.bindFramebuffer(gl.FRAMEBUFFER, null);
//send index 0 to fragment shader
        gl.uniform1i(gl.getUniformLocation(program, "i"), 0);
//normal render
        gl.clear( gl.COLOR_BUFFER_BIT );
        gl.uniform3fv(thetaLoc, theta);
        gl.drawArrays(gl.TRIANGLES, 0, 36);
    });
```

Picking by Selection

- Possible with render-to-texture
- When mouse clicked do a off screen rendering with new viewing conditions that render only a small area around mouse
- Keep track of what gets rendered to this off screen buffer