Institute for Advanced Study

Exoplanets, Milky Way and OpenGL

Hanno Rein @ NY iOS Developer Meetup

self = [super init];

Astronomer

- PhD in Cambridge, UK
- Member at the Institute for Advanced Study in Princeton
- Research interest are exoplanets, planetary rings, numerical methods, large scale computing

iOS Developer

- Sole developer of the Exoplanet App
- Outreach project
- ~ ten million downloads
- ~ one million active users

Agenda for tonight

What I will talk about

- Using large datasets such as the Hipparcos Catalogue, 2MASS Redshift Survey and the Open Exoplanet Catalogue
- Anatomy of the OpenGL ES 2.0 pipeline
- Introduction to OpenGL jargon such as vertices, attributes, primitives, shader and framebuffers
- Walk through several basic shaders
- Dust layers in our Milky Way
- Walk through a more complicated shader with multiple rendering steps

What I won't talk about

- How to setup an OpenGL view in an iPhone application. *Look at the Xcode example and the documentation provided by Apple. You'll only need to do this once.*
- Matrices, vectors and matrix operations. *Dig out your high school algebra book. There are also many articles about this on the internet. Use the GLKit functions whenever possible.*
- How to compile shaders and upload textures. *Copy and paste your favorite snippet from the web. You'll only need to do this once.*
- Thousands of OpenGL pitfalls. *Stackoverflow is your friend.*

Large datasets

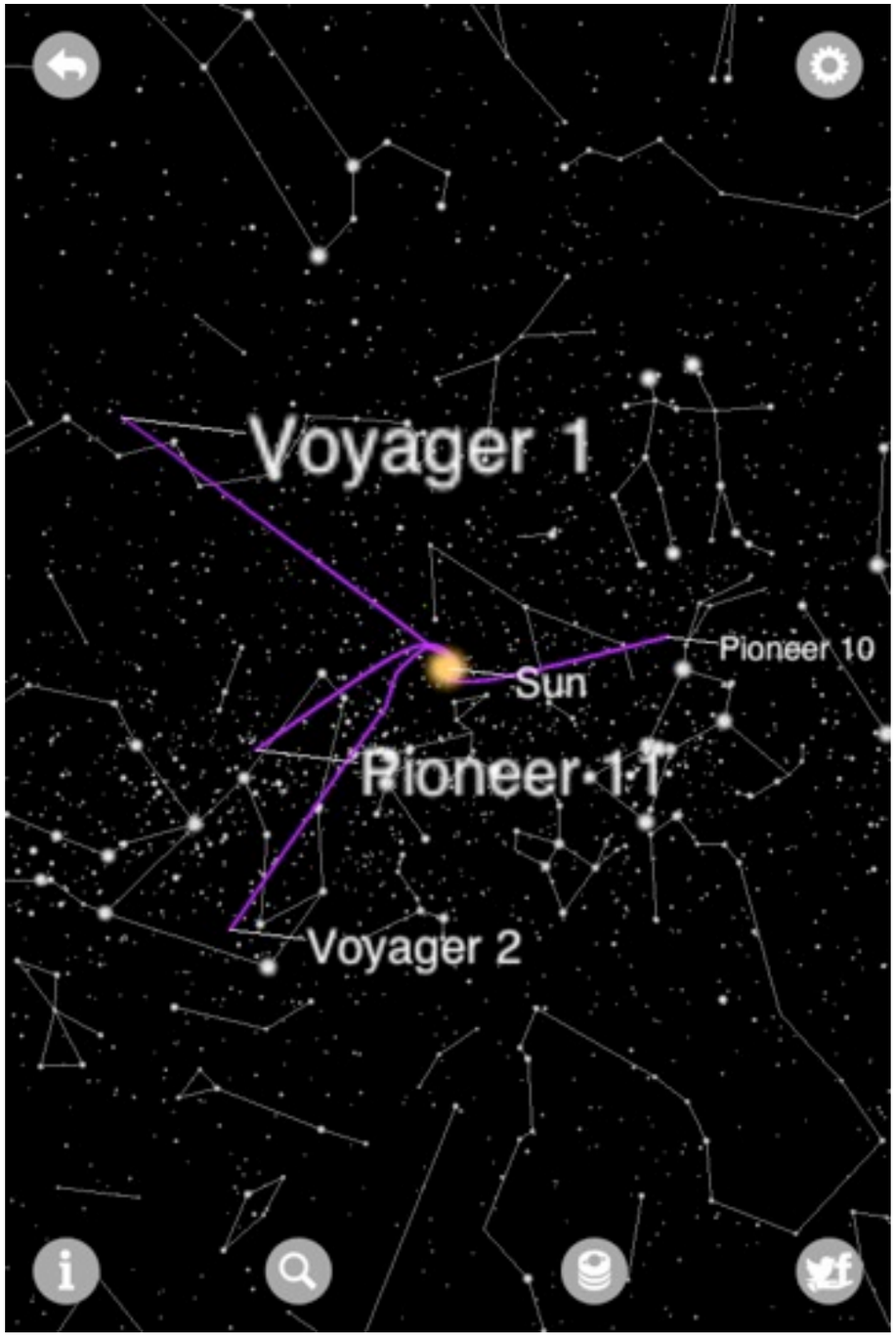
Demo*

* fingers crossed. Screenshot created with PlaceIt.

Large datasets used in the Exoplanet App

Spacecrafts

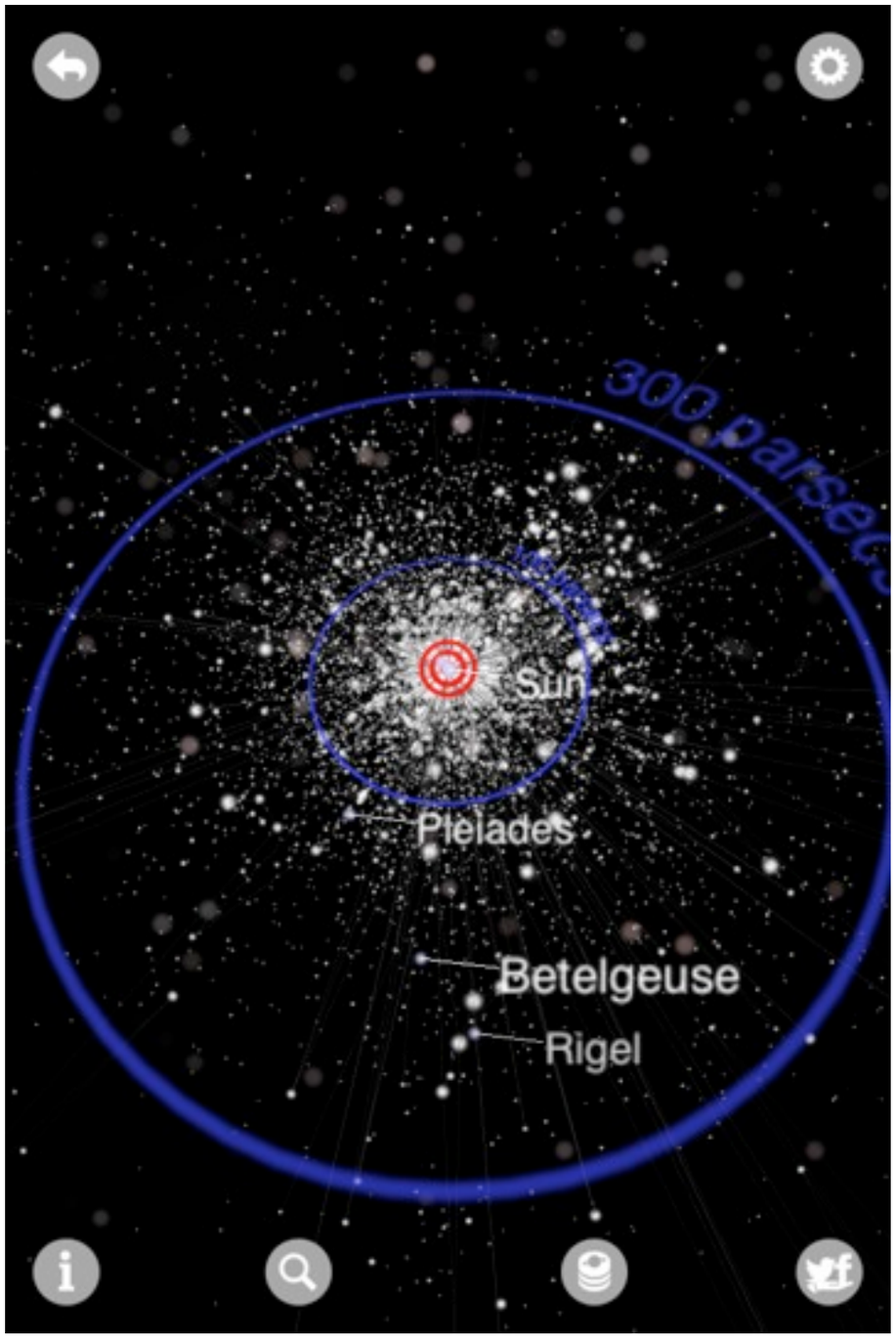
- 4 spacecrafts
- 32768 points
- Position only



Large datasets used in the Exoplanet App

Hipparcos Catalogue

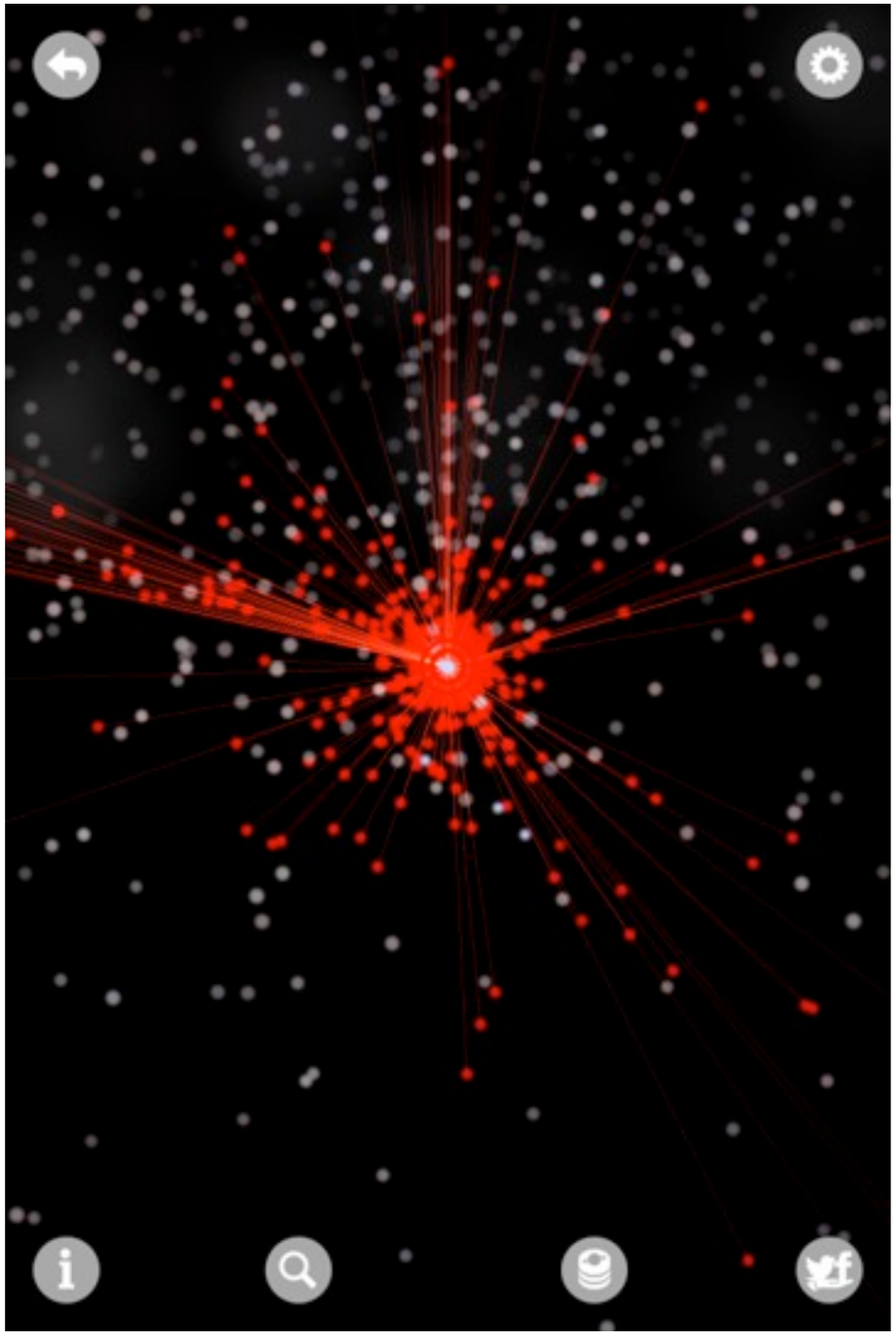
- 34904 nearby stars
- Position and brightness
- Linked to constellation lines



Large datasets used in the Exoplanet App

Exoplanets

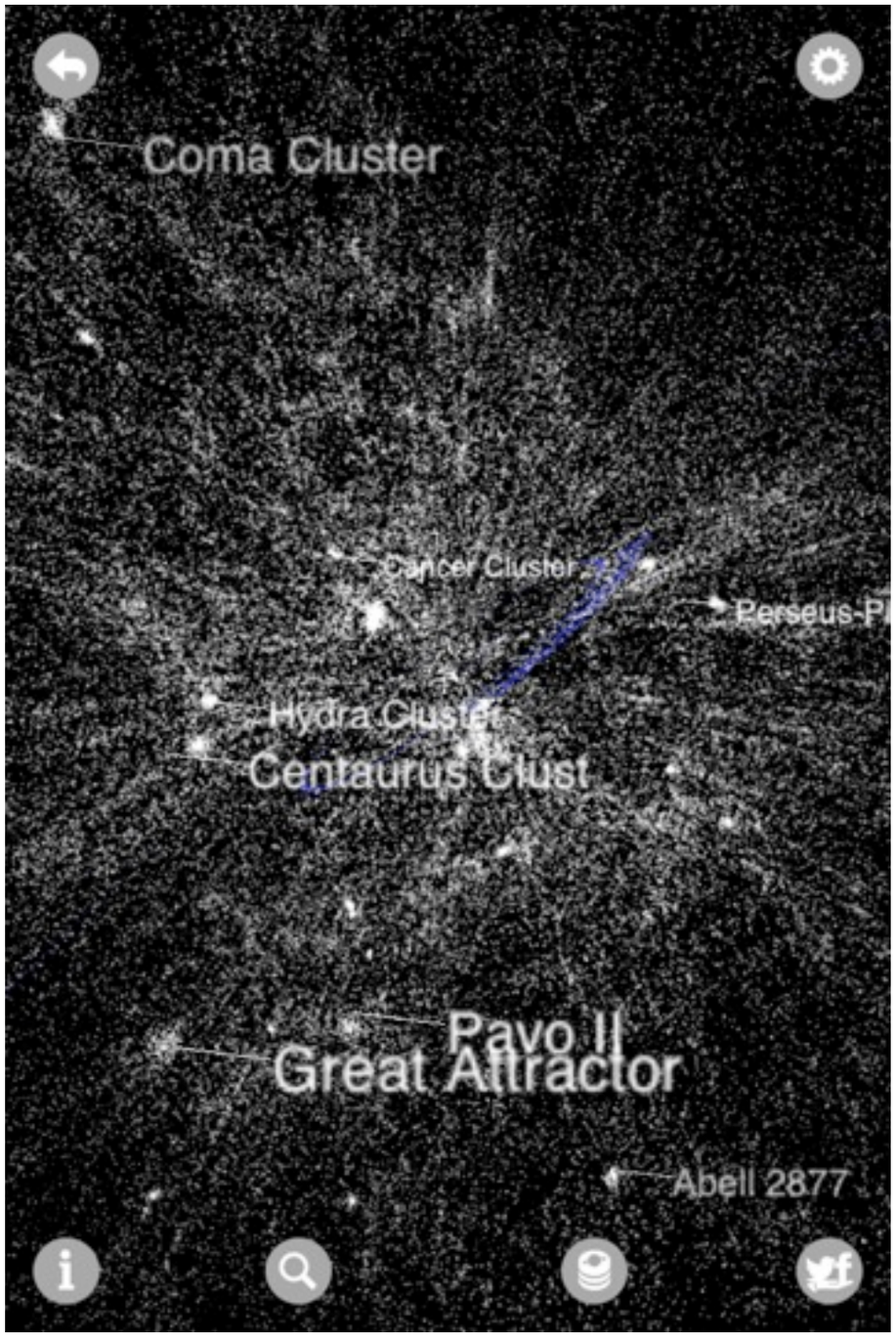
- 864 planets
- Lots of meta-data
- Maintained by myself
- Used for my own research



Large datasets used in the Exoplanet App

2MASS Redshift survey

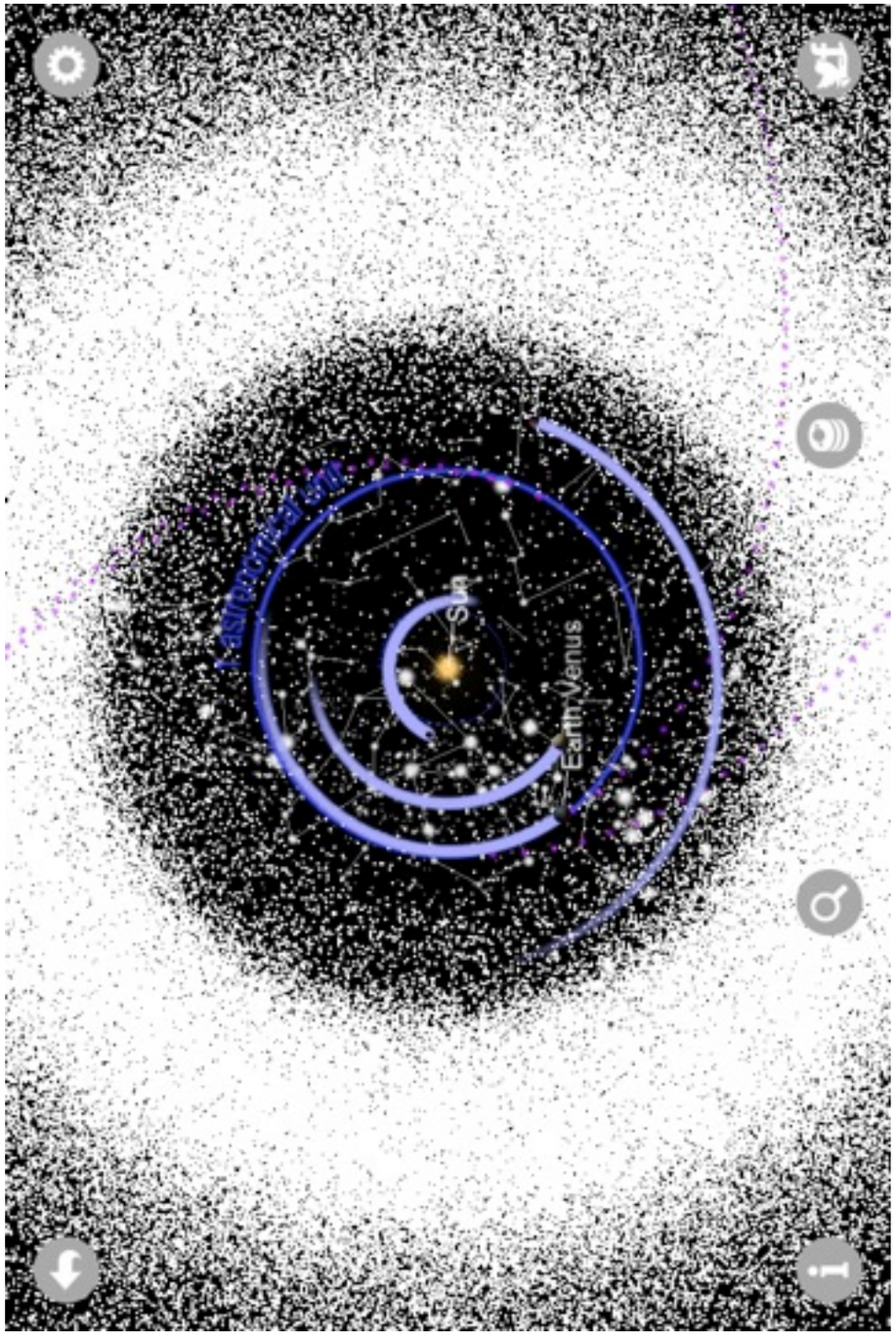
- 1173144 galaxies and quasars in the cosmic neighborhood
- Position only
- Close to the maximum that the iPhone can render easily



Large datasets used in the Exoplanet App

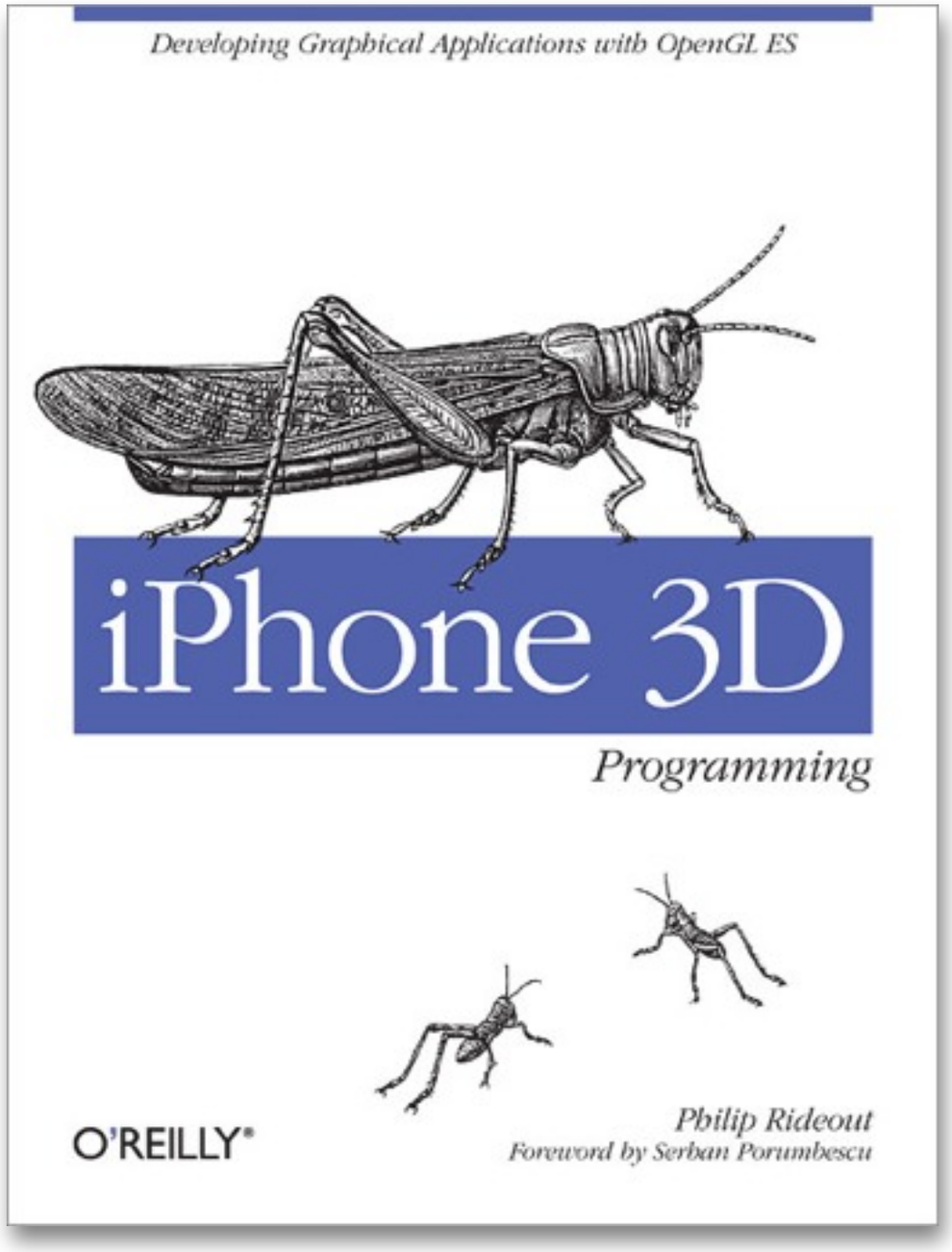
Asteroids

- 320,000 Asteroids
- Position only, need to be dynamically integrated to account for Jupiter's gravity



An astronomer's introduction to OpenGL

Further reading

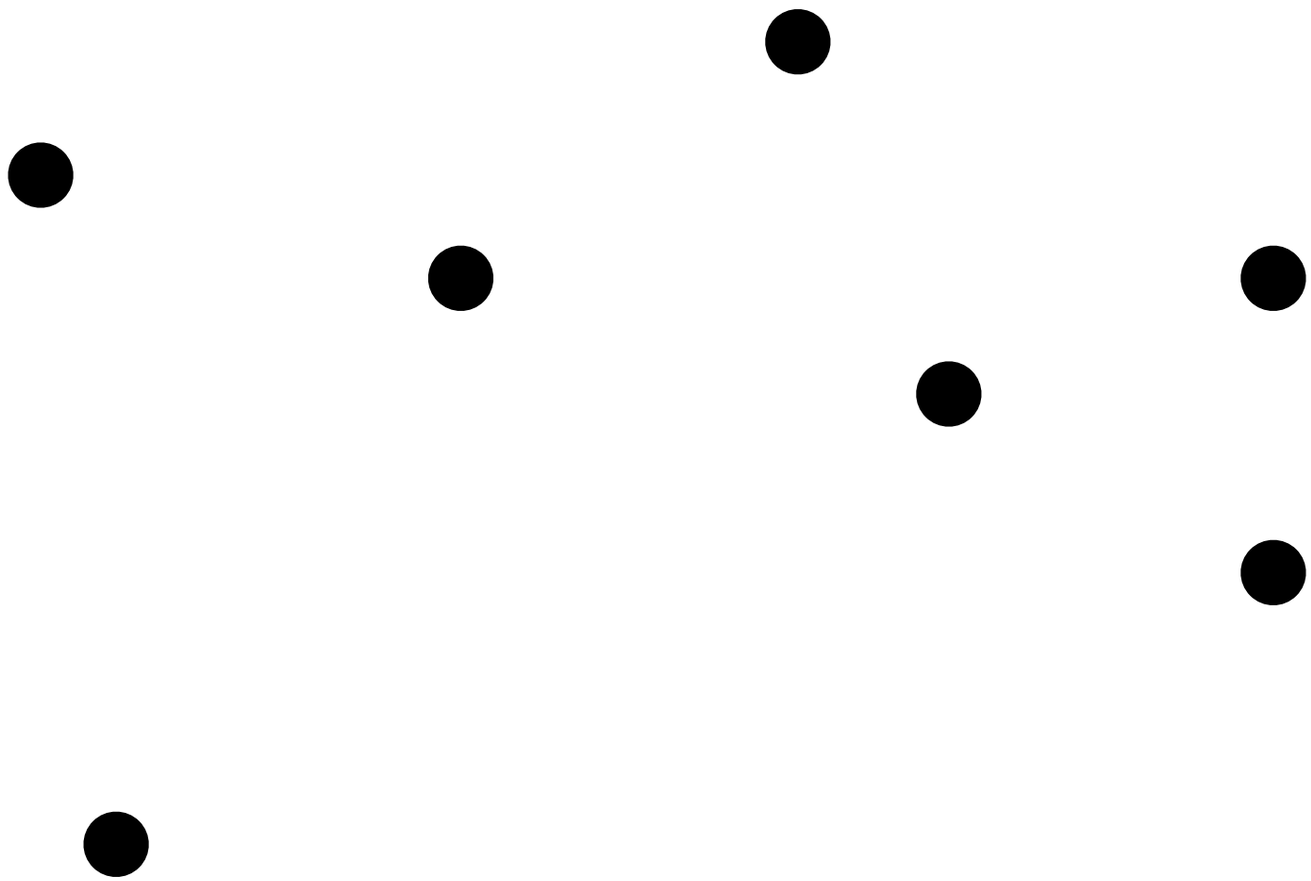


iPhone 3D Programming by Philip Rideout, O'Reilly Media

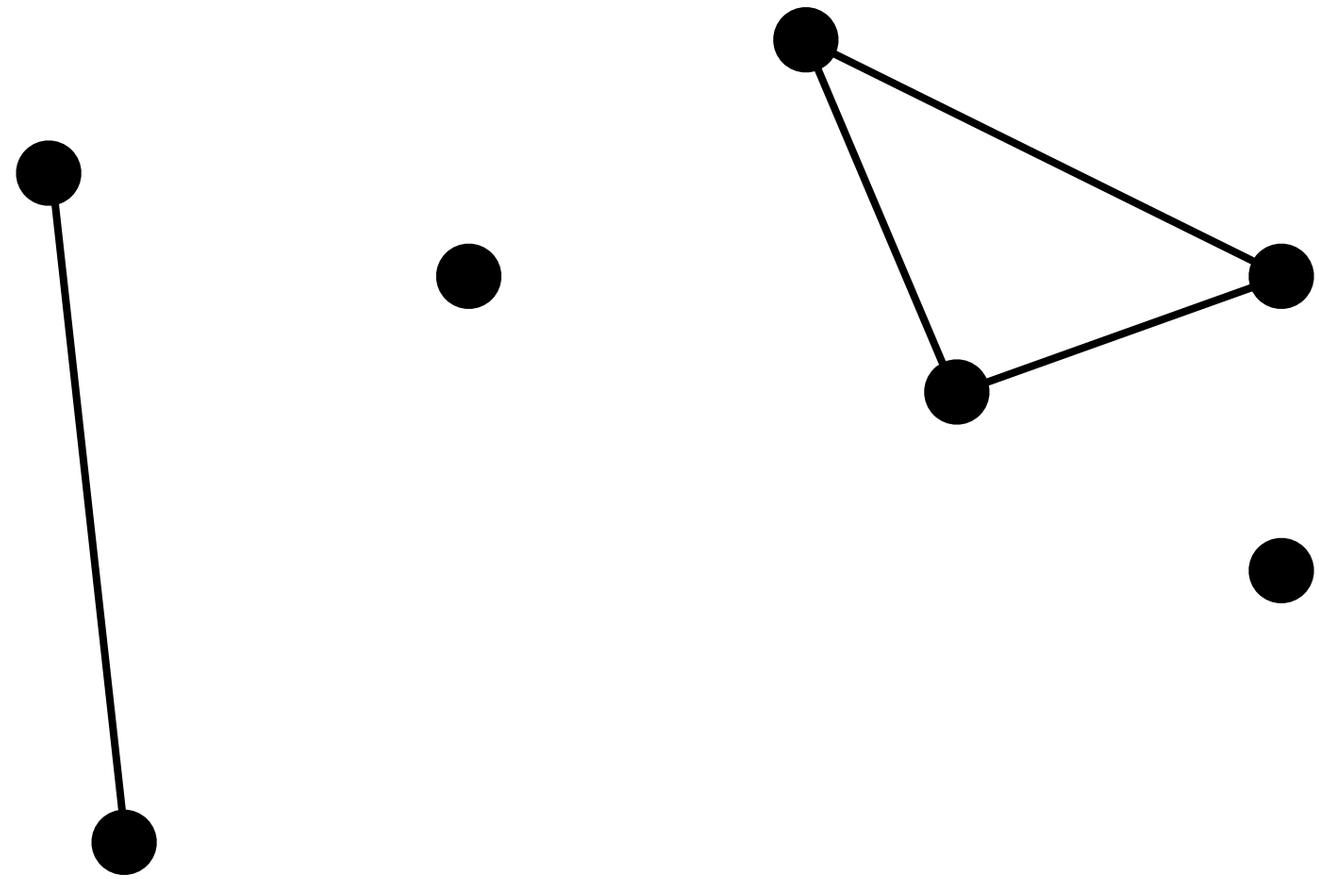
It's a good time to start ignoring OpenGL ES 1.1

Device	OpenGL ES 2.0
iPod Touch	No
iPod Touch (Second Generation)	No
iPod Touch (Third Generation)	Yes
iPod Touch (Fourth Generation)	Yes
iPod Touch (Fifth Generation)	Yes
iPhone	No
Phone 3G	No
iPhone 3GS	Yes
iPhone 4	Yes
iPhone 4S	Yes
iPhone 5	Yes
iPad (All Generations)	Yes

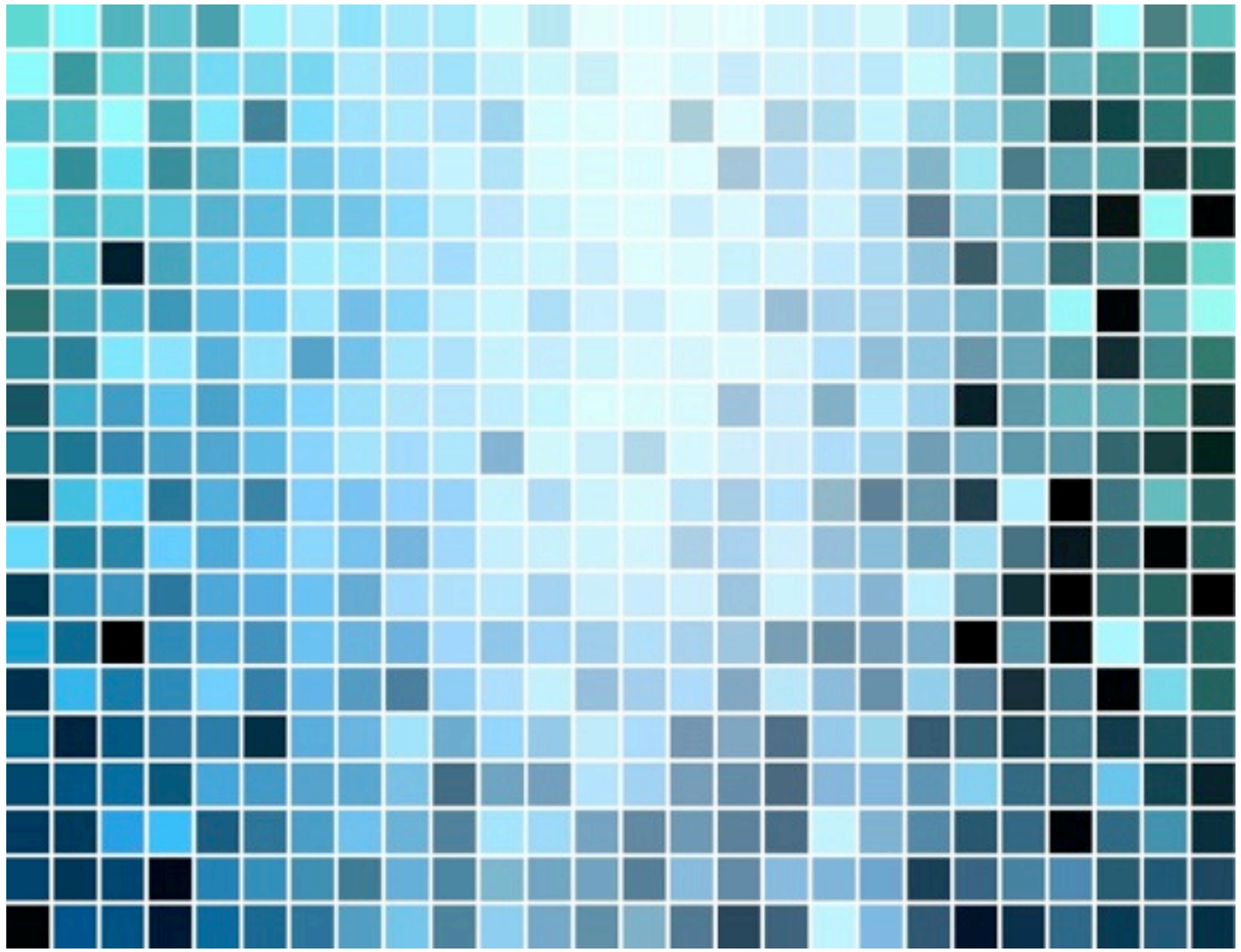
Vertices



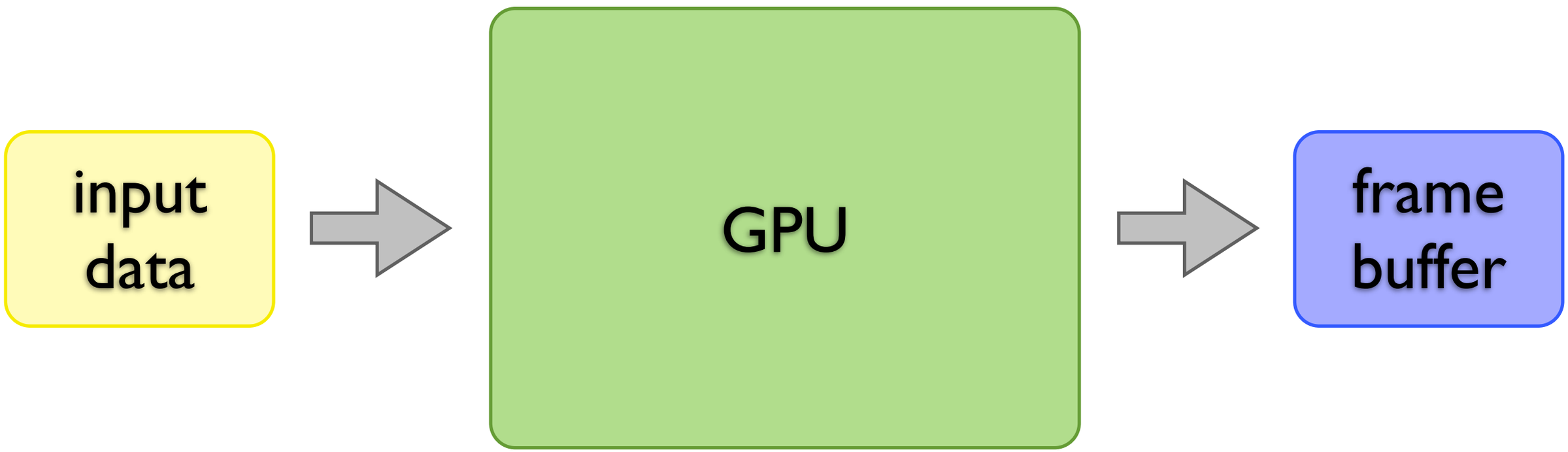
Primitives



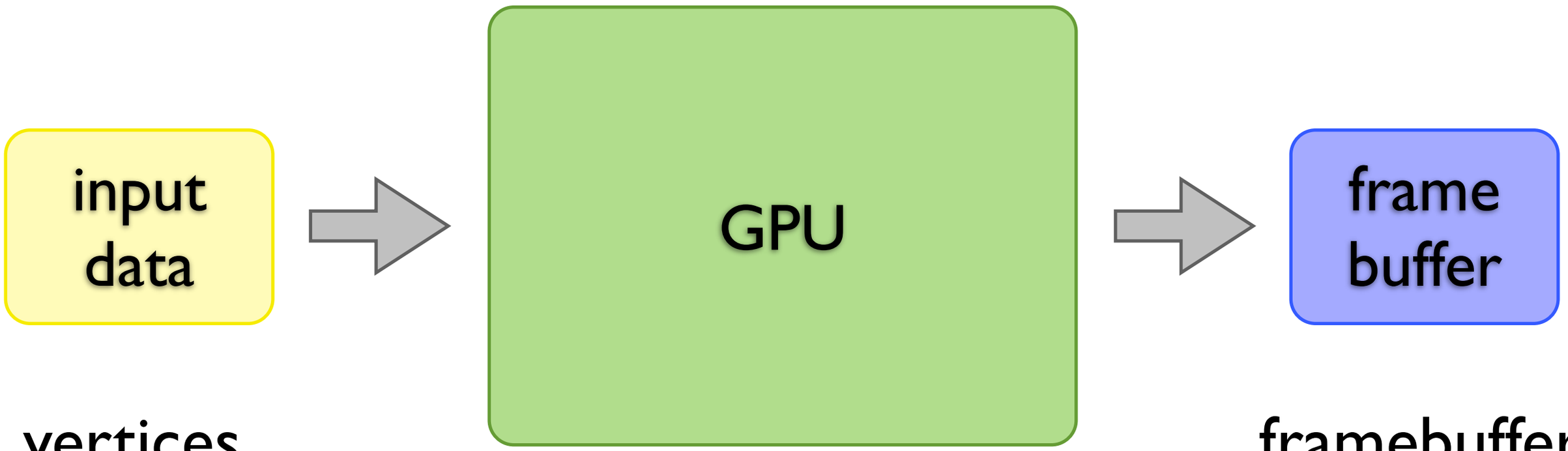
Pixels, framebuffers



What OpenGL actually does



Basic OpenGL jargon

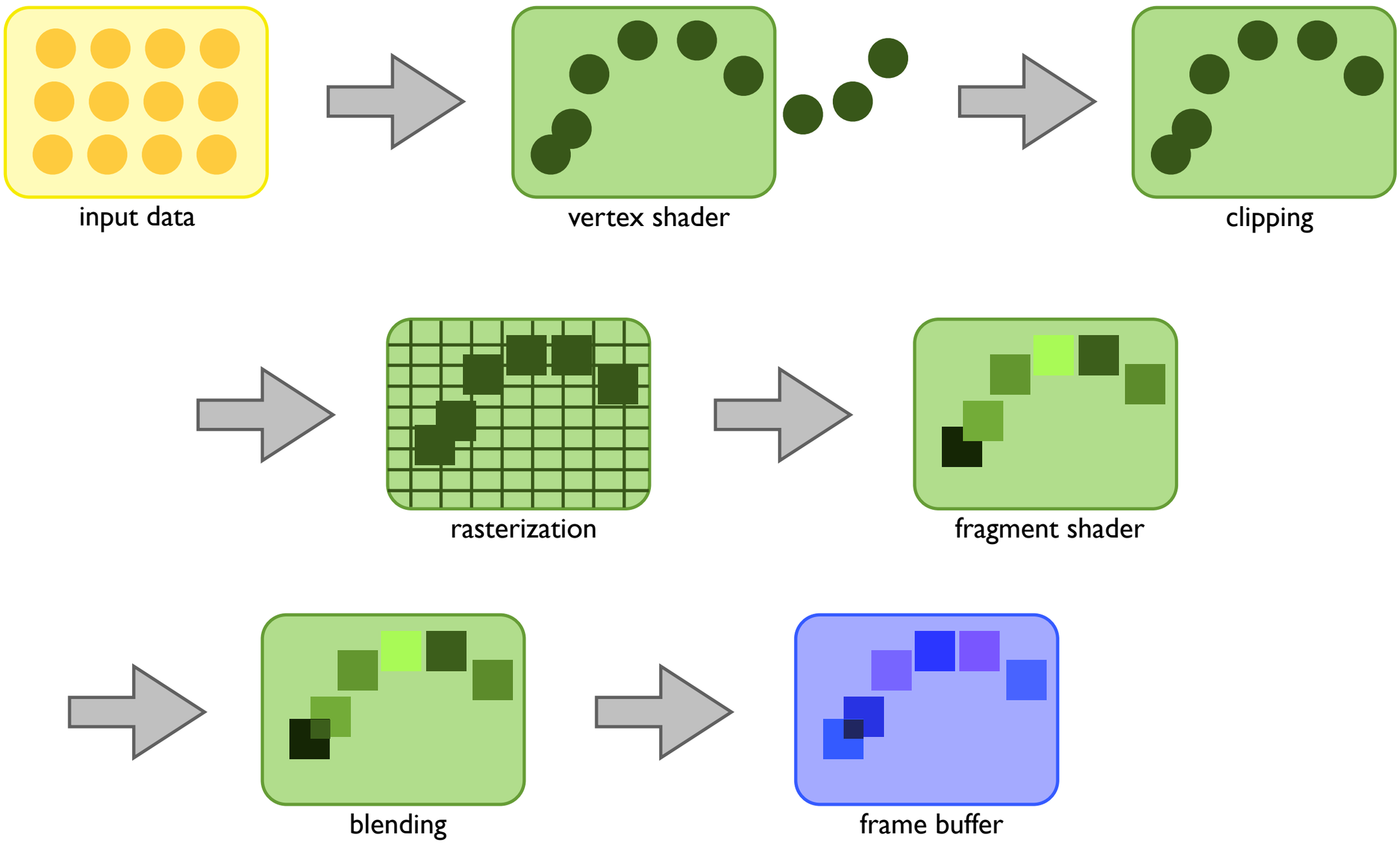


vertices,
attributes,
uniforms,
textures

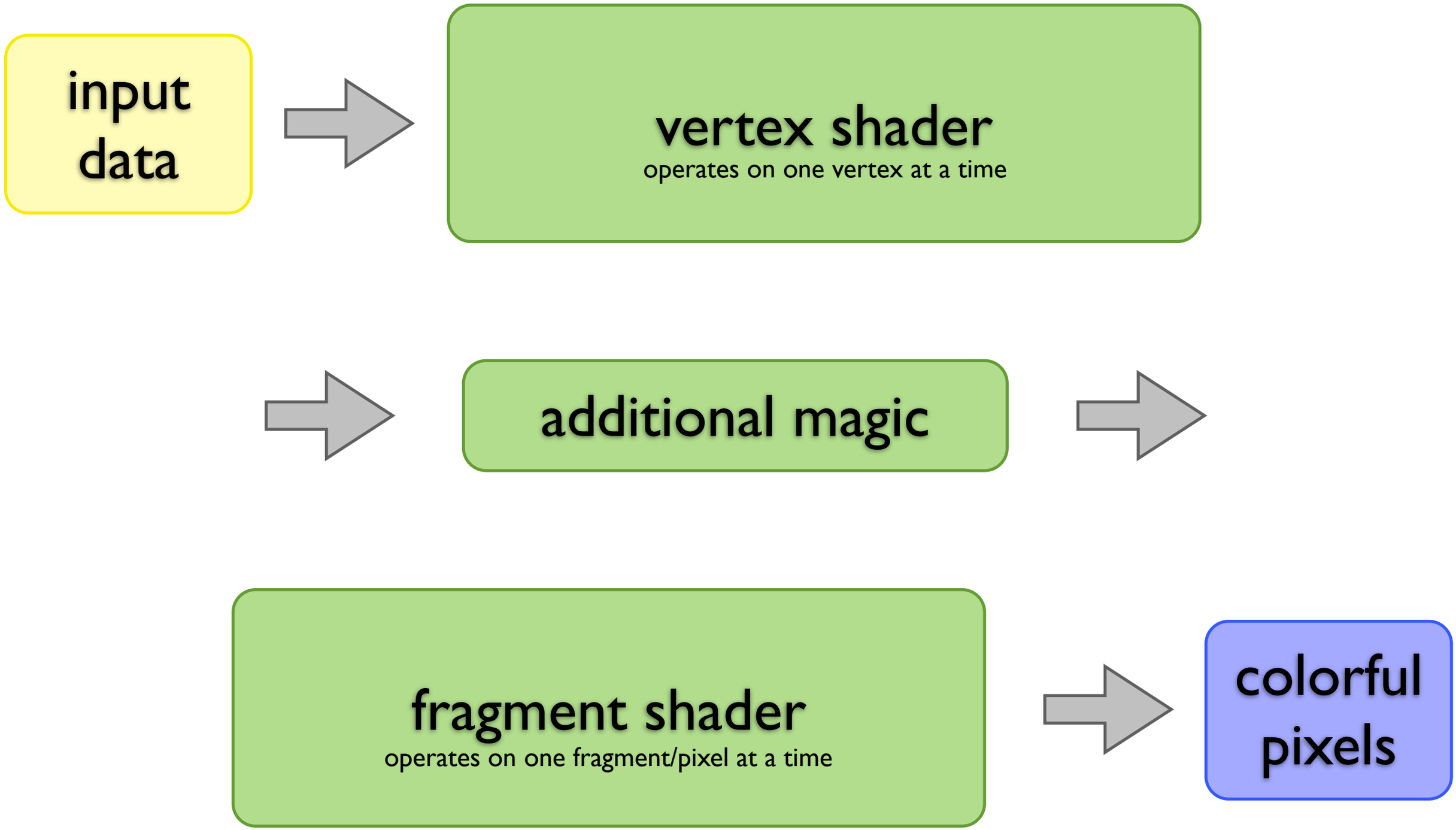
geometry shaders,
vertex shaders,
fragment shaders

framebuffer,
colorbuffer,
depthbuffer,
stencilbuffer

OpenGL ES 2.0 pipeline



Shaders


input data
vertex shader
operates on one vertex at a time
additional magic
fragment shader
operates on one fragment/pixel at a time
colorful pixels

Vertex shader

input data

one vertex at a time

can use attributes
can use uniforms

vertex shader

What you do in here is entirely up to you.

Must write to

```
highp   vec4  gl_Position;
```

May write to

```
mediump float gl_PointSize;
```

Input data for the vertex shader

constant
storage qualifier

```
const vec3 zAxis = vec3 (0.0, 0.0, 1.0);
```

- compile time constant

uniform
storage qualifier

```
uniform vec4 lightPosition;
```

- global variable
- the same across all vertices
- read-only

attribute
storage qualifier

```
attribute vec4 position;
```

- different value for every vertex
- read-only

Magic

er

`gl_Position, gl_PointSize`
for every vertex

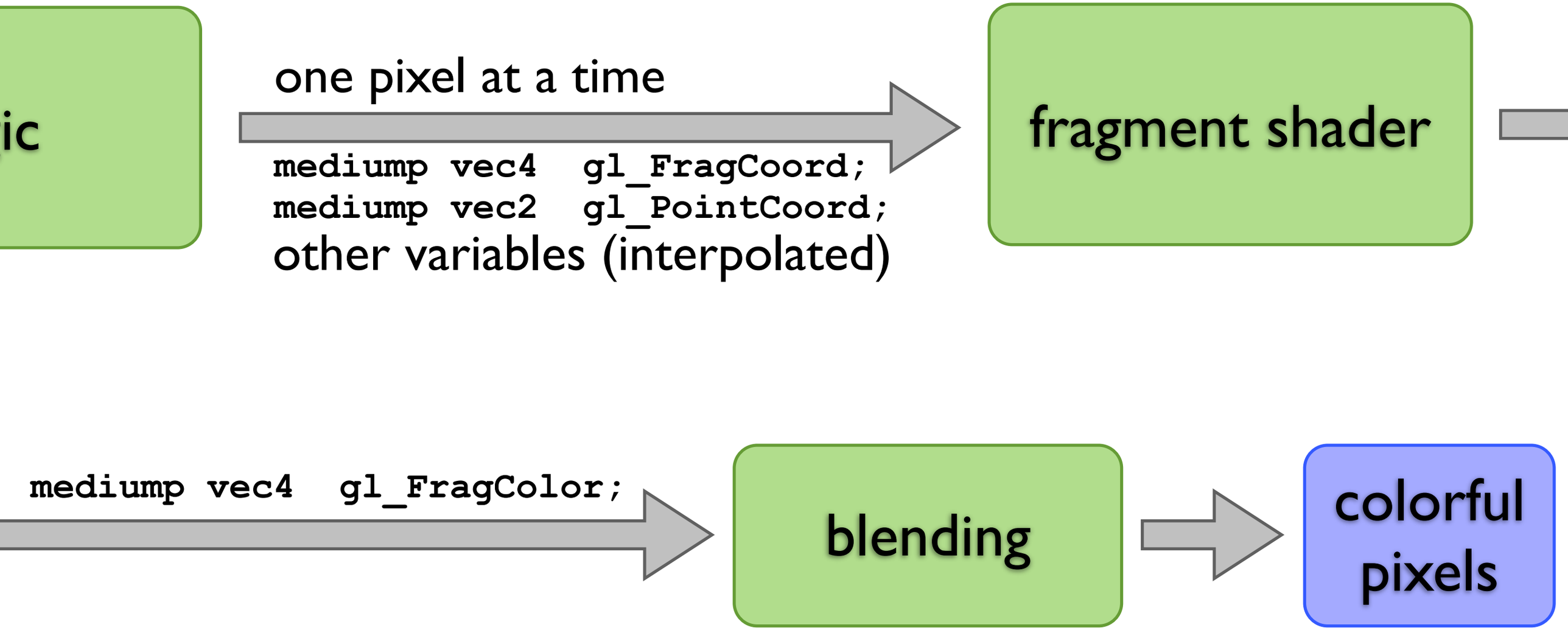
various other variables set
by API, e.g. primitive type,
dimensions of the canvas, ...

primitive assembly

rasterization

clipping
culling

Fragment shader



A first look at an actual shader

Generating input data

```
float* data = malloc(sizeof(float)*numStars*3);

for (int i=0; i<numStars; i++) {
      data[i*3+0] = (float)rand()/(float)RAND_MAX;
      data[i*3+1] = (float)rand()/(float)RAND_MAX;
      data[i*3+2] = 1.;
}
```

```
glGenBuffers(1, &(vertexBuffer));
glBindBuffer(GL_ARRAY_BUFFER, vertexBuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(float)*numStars*3,
   data, GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);

free(data);
```

Simplest possible shader

Vertex shader

```
attribute vec4 Position;

void main(){
  gl_Position = Position;
  gl_PointSize = 10.;
}
```

Fragment shader

```
void main(){
  gl_FragColor = vec4(1.,1.,1.,1.);
}
```

Calling code

```
SimpleShader* shader =
      [ShaderManager getSimpleShaderWithName:@"HelloWorld"];
glUseProgram(shader.program);



glBindBuffer(GL_ARRAY_BUFFER, vertexBuffer);
glVertexAttribPointer([shader getAttribute:@"Position"],
  3, GL_FLOAT, GL_FALSE, sizeof(float)*3, 0);
glEnableVertexAttribArray([shader getAttribute:@"Position"]);



glDrawArrays(GL_POINTS, 0, numStars);



glBindBuffer(GL_ARRAY_BUFFER, 0);
glDisableVertexAttribArray([shader getAttribute:@"Position"]);
```

Result

- 128 points
- 10 pixel size
- All points are in top right quadrant
- Coordinate system ranges from [-1:1] but we used [0:1]
- Model-View-Projection matrix is a complicated way to transform your points to [-1:1]

Doing a calculation on the GPU

Doing work...

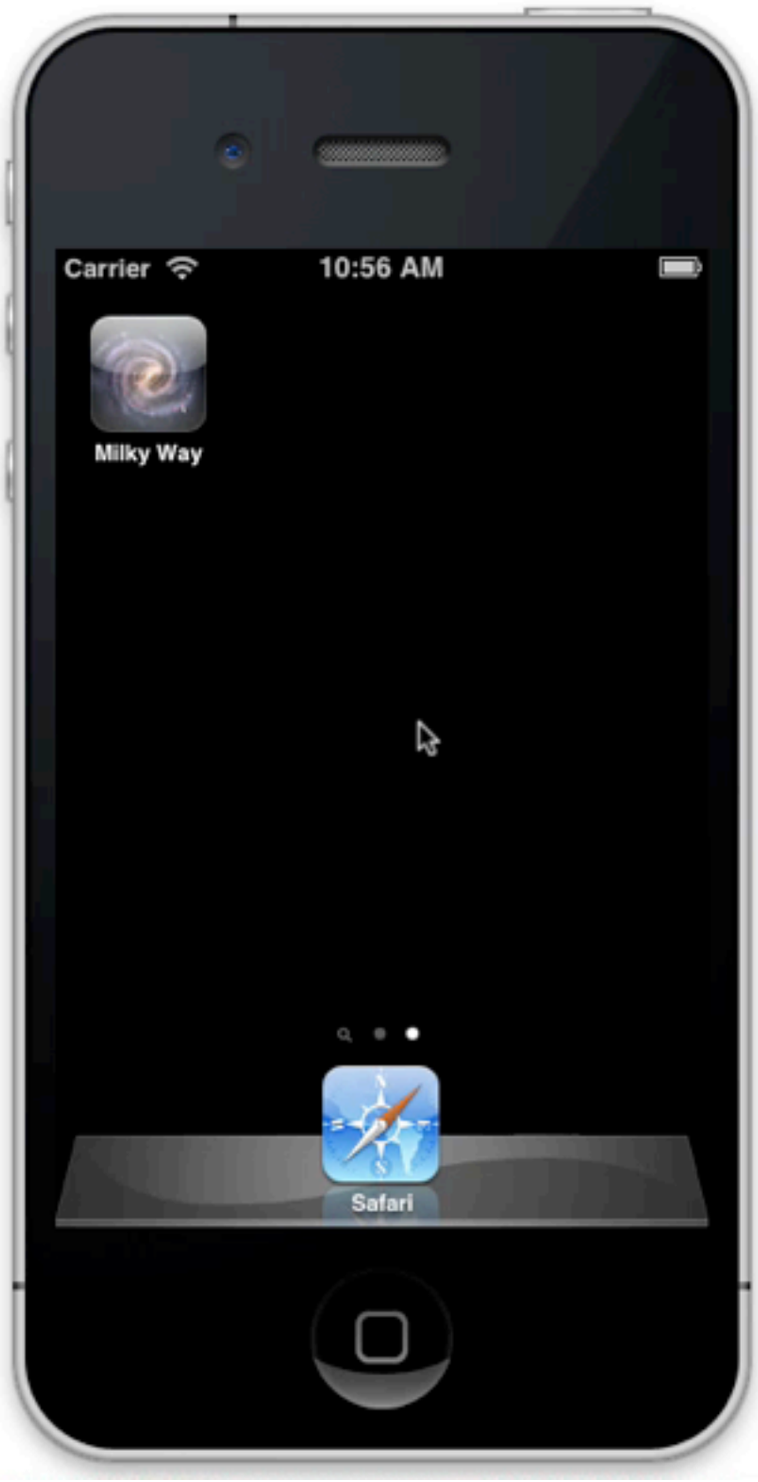
Vertex shader

```
attribute vec4 Position;
uniform float Time;

void main(){
    float radius = length(Position.xy);

    float rot_x = Position.x * sin(Time*radius)
                + Position.y * cos(Time*radius);
    float rot_y = Position.x * cos(Time*radius)
                - Position.y * sin(Time*radius);

    gl_Position = Position;
    gl_Position.x = rot_x;
    gl_Position.y = rot_y;

    gl_PointSize = 10.;
}
```

Calling code

```
...

glUniform1f([shader getUniform:@"Time"], time);
time += 0.1;

...
```

Result

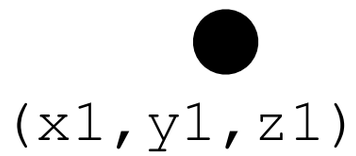


Other primitives

0D, 1D, 2D primitives exist but 3D is a lie

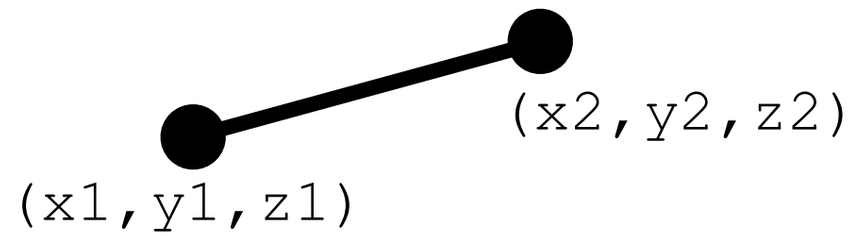
GL_POINTS

primitive



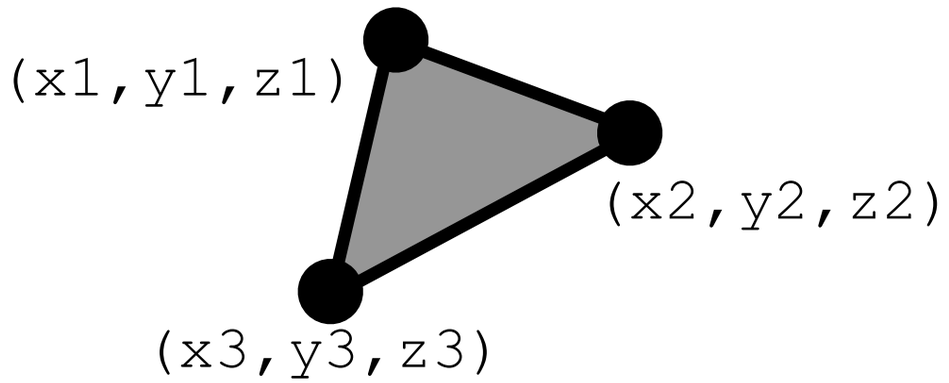
GL_LINES

primitive



GL_TRIANGLES

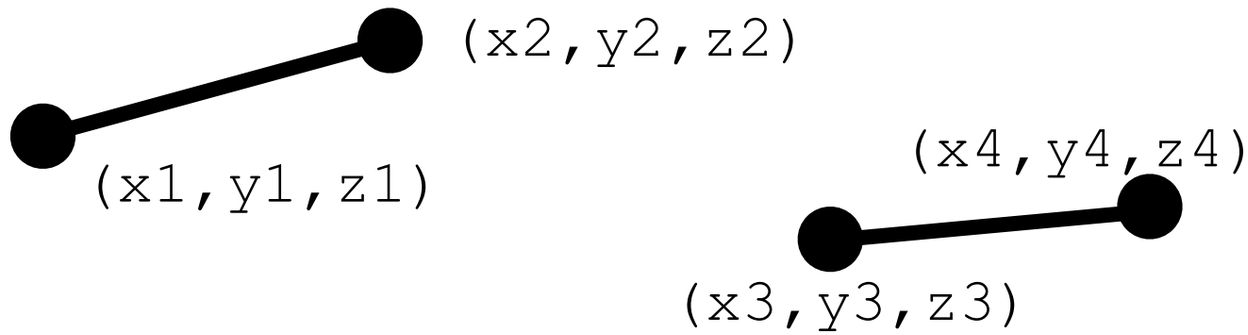
primitive



1D primitives

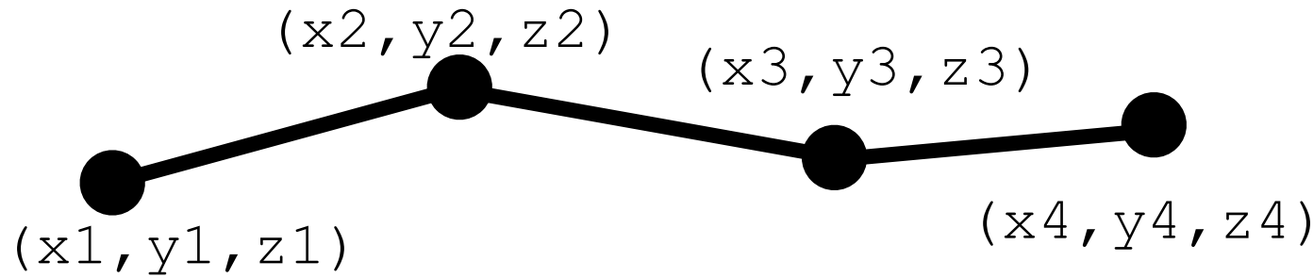
GL_LINES

primitive



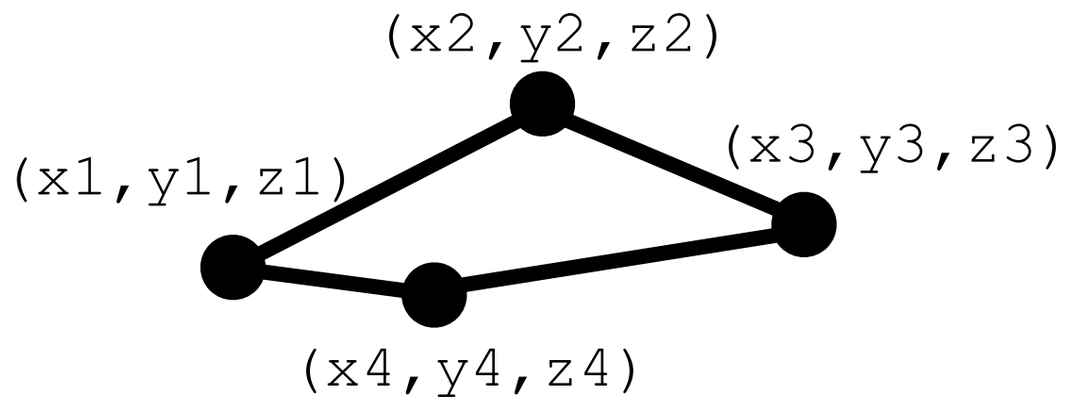
GL_LINE_STRIP

primitive



GL_LINE_LOOP

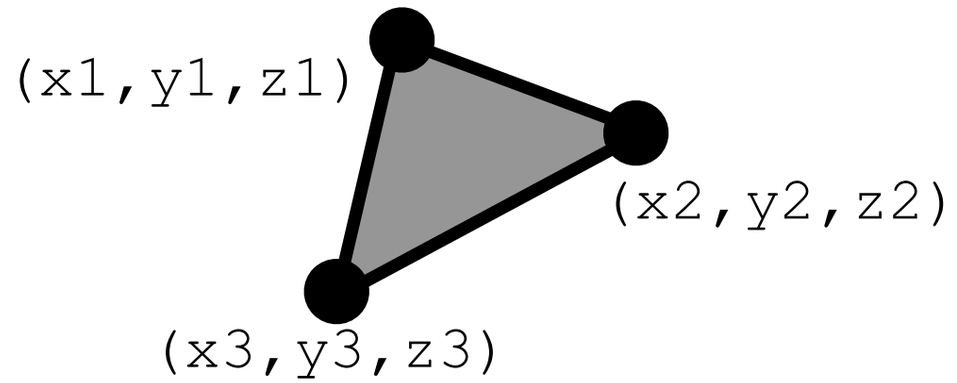
primitive



2D primitives

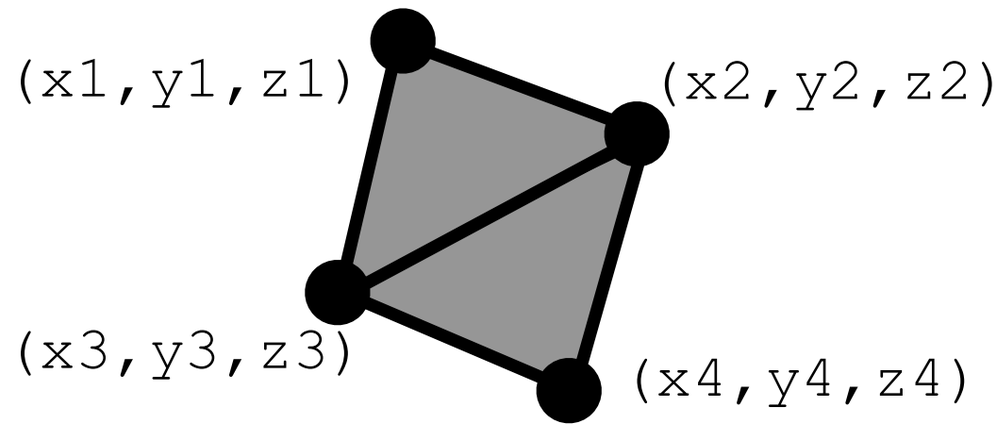
GL_TRIANGLES

primitive



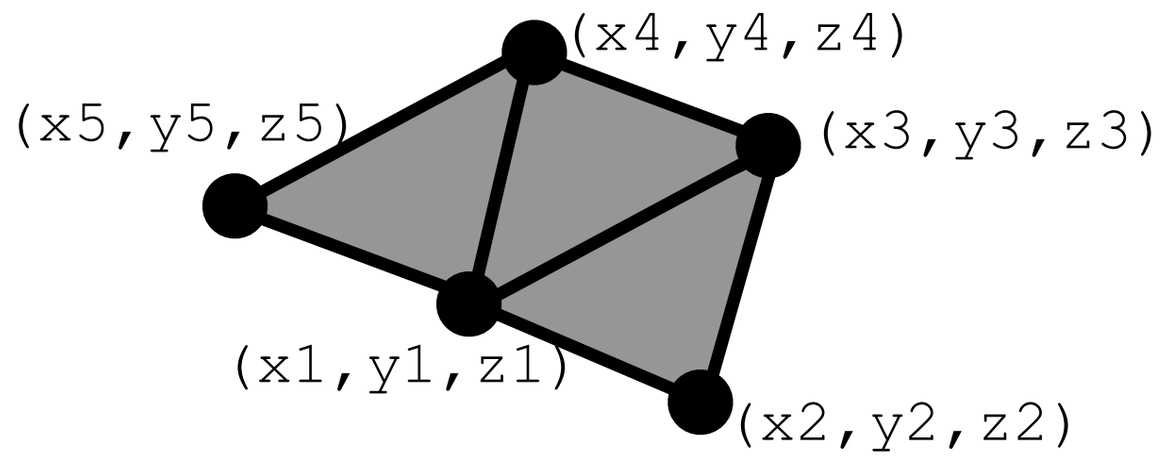
GL_TRIANGLE_STRIP

primitive



GL_TRIANGLE_FAN

primitive

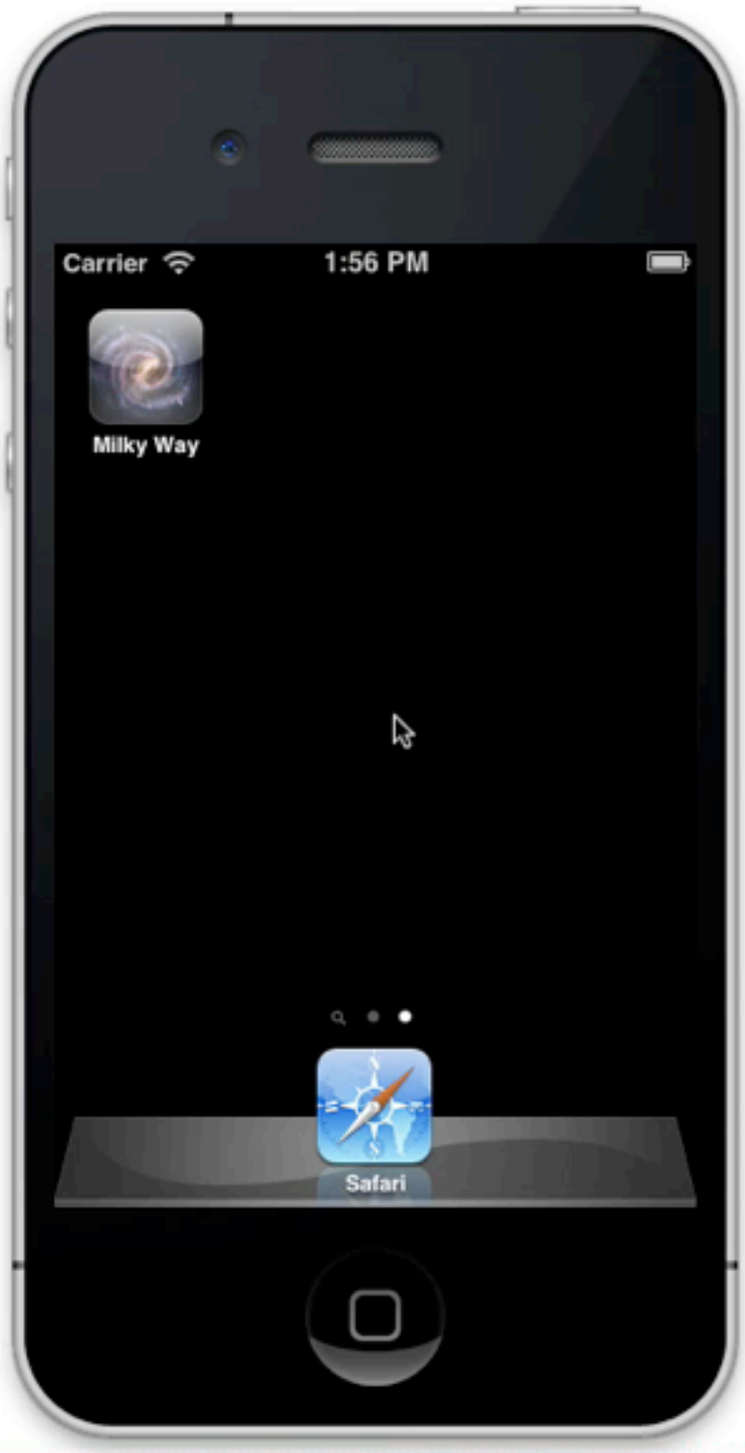


Drawing lines instead of points

Calling code

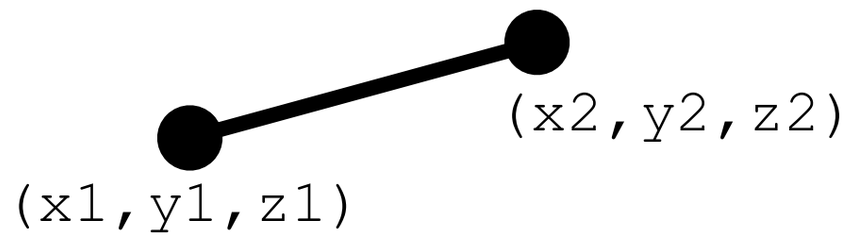
```
...
glDrawArrays(GL_POINTS, 0, numStars);
glDrawArrays(GL_LINES, 0, numStars);
...
```

Result



GL_LINES

primitive



Adding color and the varying keyword

One more variable type

varying
storage qualifier

```
varying lowp vec3 color;
```

- Output of the vertex shader
- Input of the fragment shader
- Interpolated between vertices
- Need to specify precision qualifier

GLSL Version ≥ 1.40:

attribute storage qualifier	→	in storage qualifier (vertex shader)
varying	→	out / in

Hello Colorful World

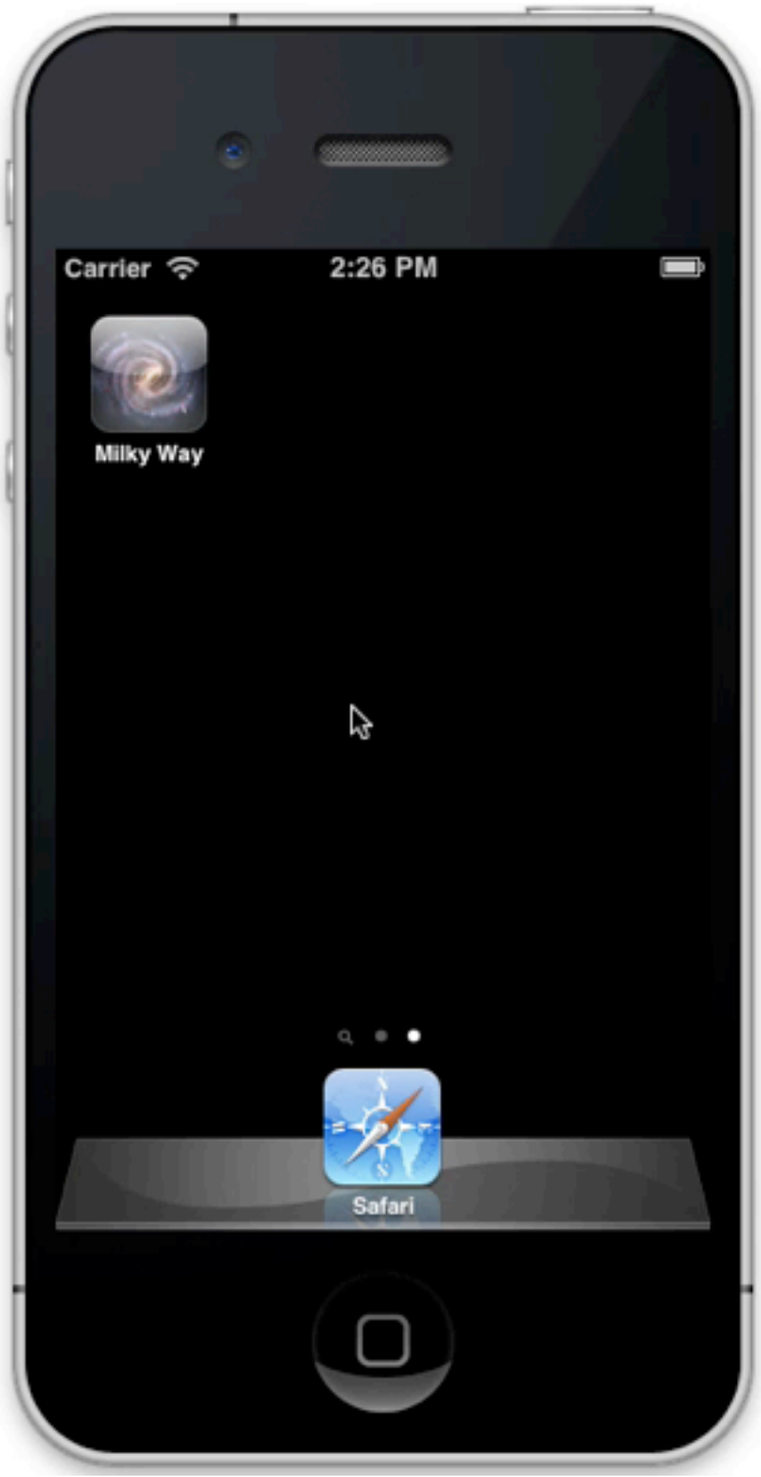
Vertex shader

```
attribute vec4 Position;
uniform float Time;
varying lowp vec3 color;

void main(){
    ....
    color.r = sin(Time)*sin(Time);
    color.g = radius;
    color.b = cos(atan(rot_x,rot_y));
    ....
}
```

Fragment shader

```
varying lowp vec3 color;

void main(){
  gl_FragColor = vec4(color,1.);
}
```

Hello Colorful World

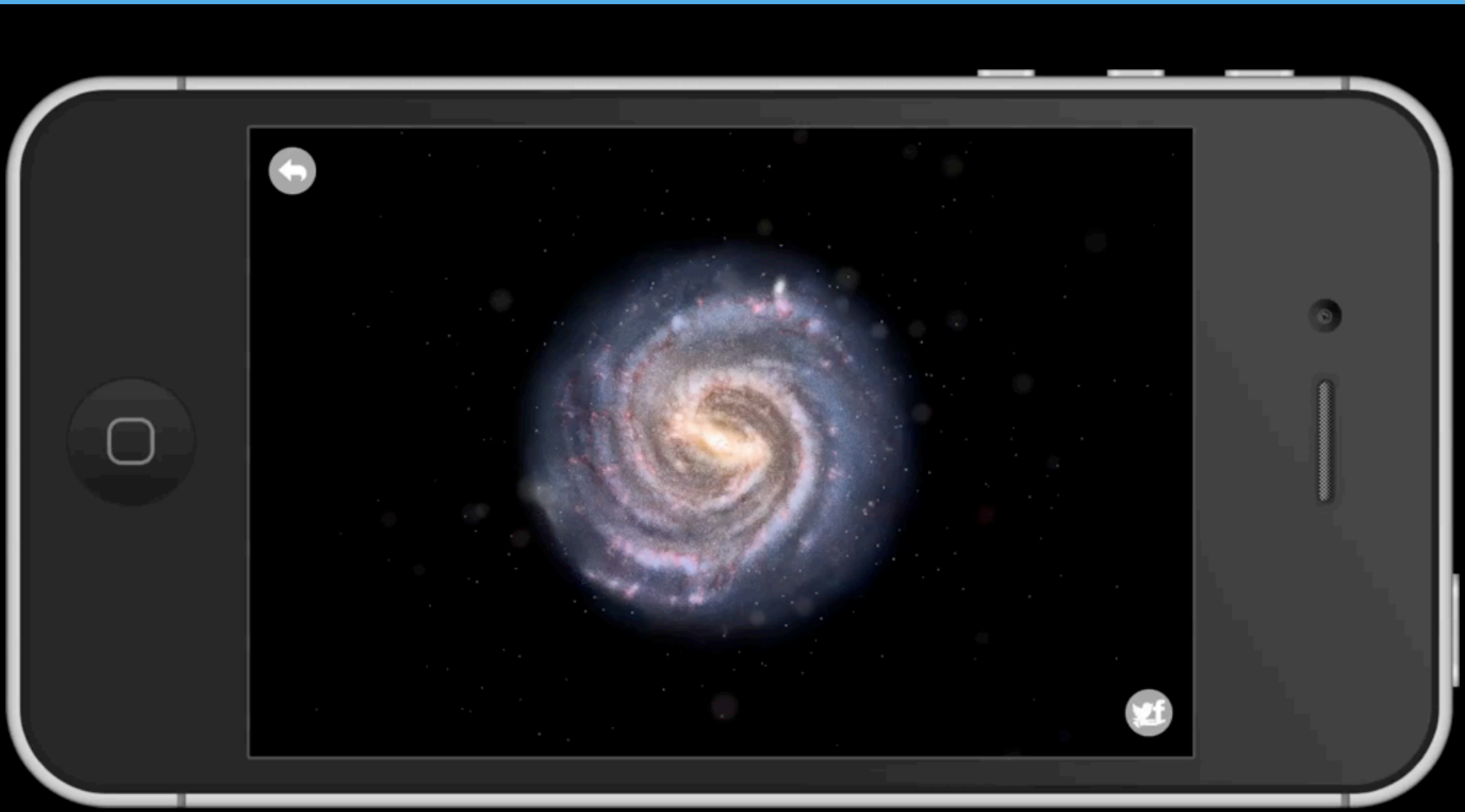


Dust in the Milky Way and a more complicated shader

GPU based ray tracing using an off screen framebuffer

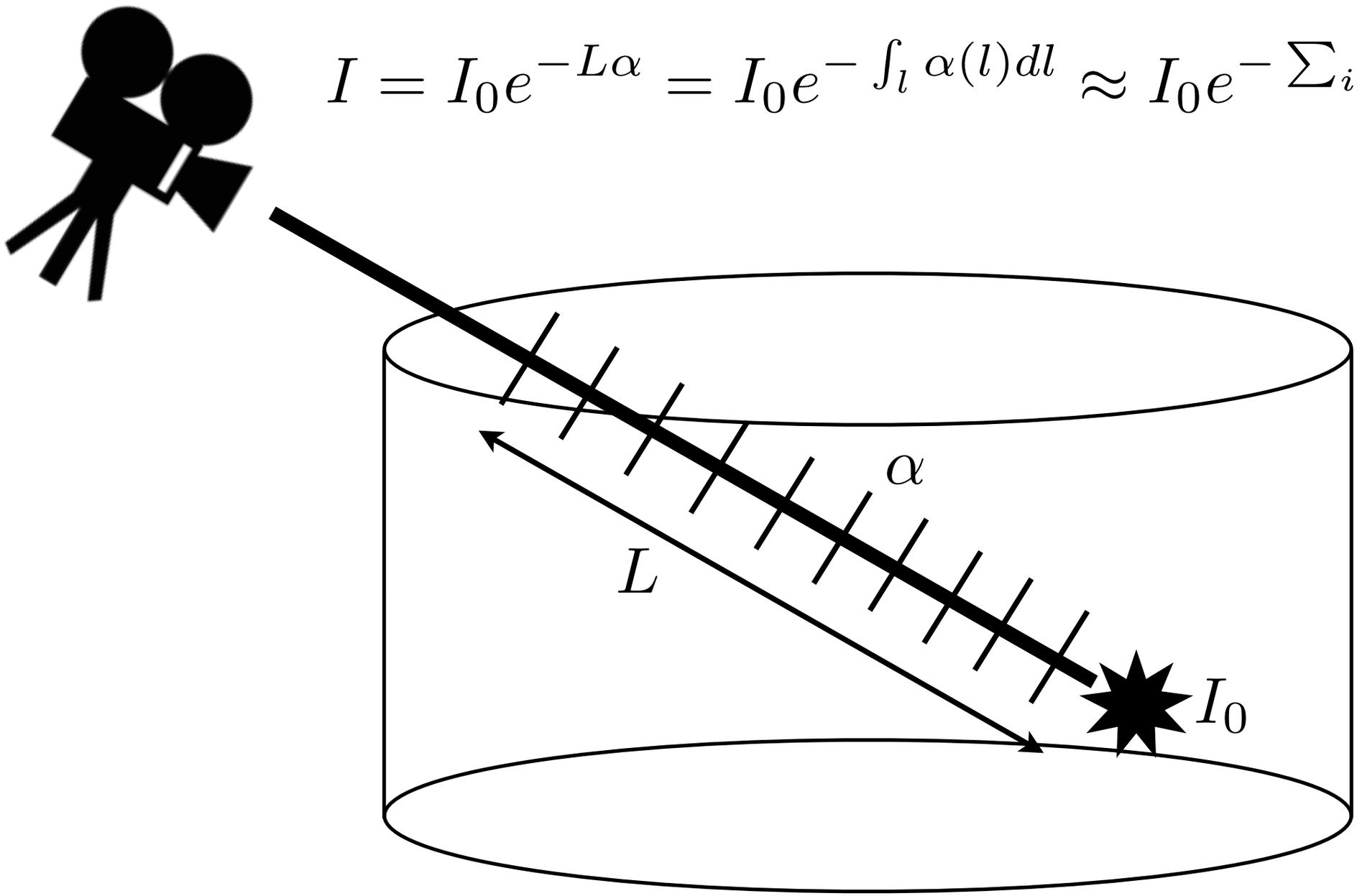
The Milky Way

The Milky Way



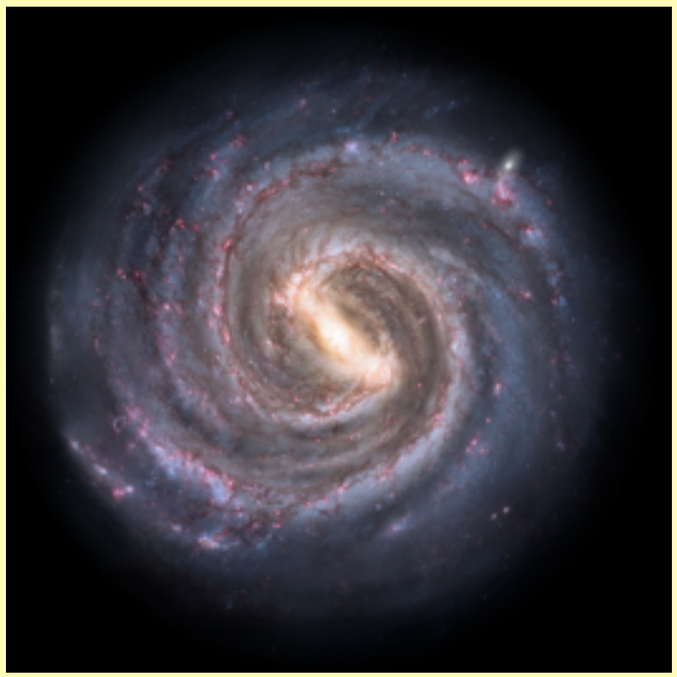
Physics

$$I = I_0 e^{-L\alpha} = I_0 e^{-\int_l \alpha(l) dl} \approx I_0 e^{-\sum_i \alpha_i}$$



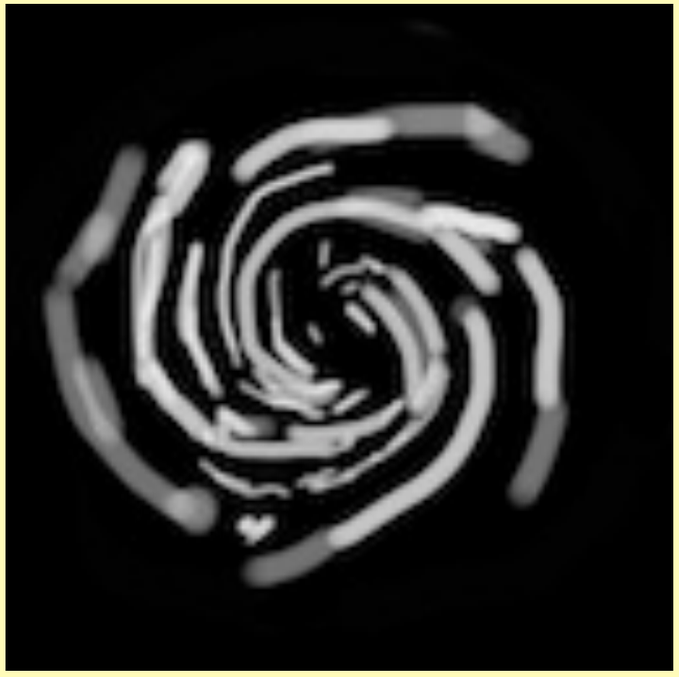
Ingredients

Background texture



1024 x 1024

Dust map



128 x 128

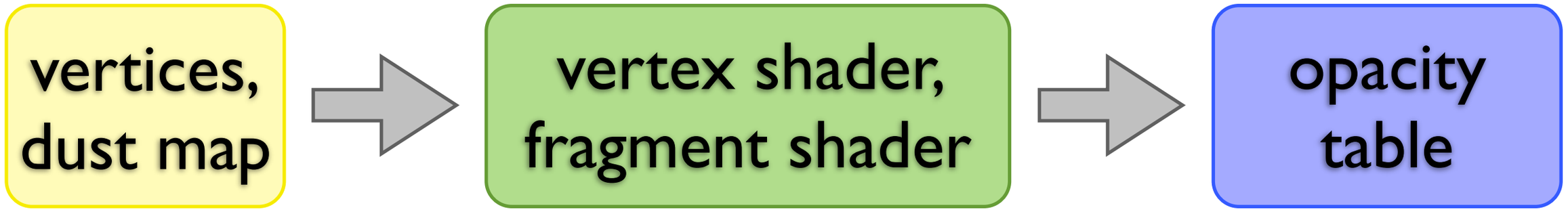
Point texture

32 x 32

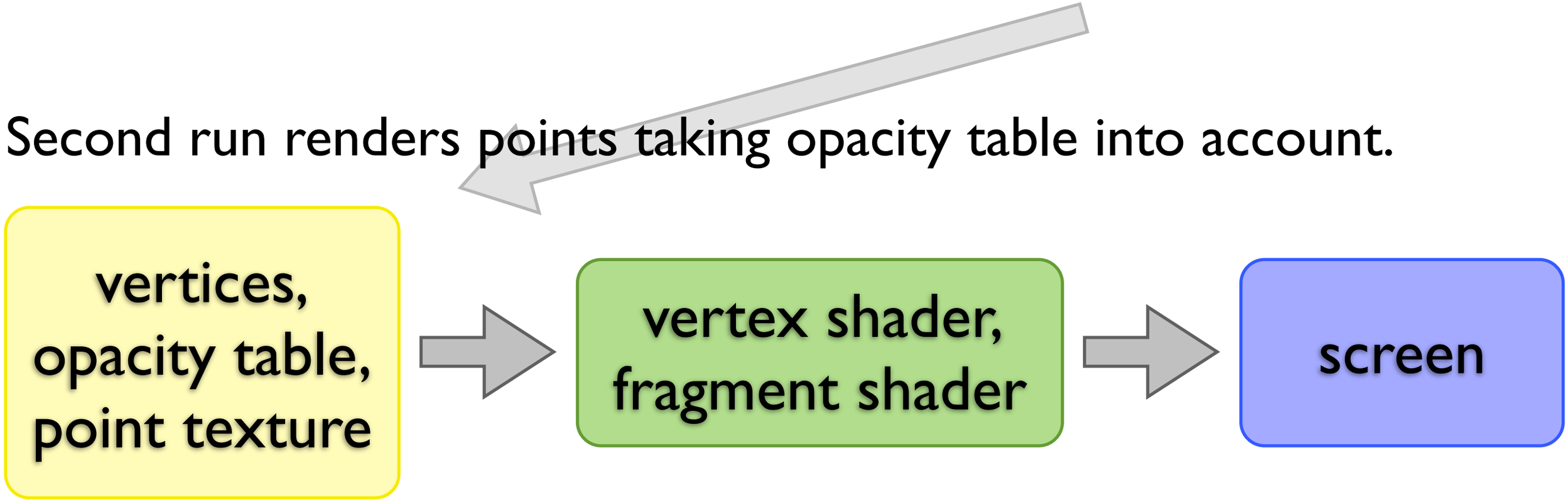
65536 vertices created by sampling background texture

Basic idea

First run calculates the integrated dust from the point source to the camera.



Second run renders points taking opacity table into account.



Render Step 1

256^2 vertices, dust map

Vertex shader

- Calculates the line of sight for every vertex
- Calculates the start and end points of the line of sight on the dust map

Fragment shader

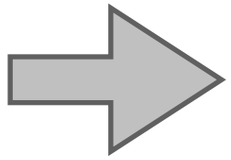
- Uses fixed number of iterations
- Integrates (sums over) dust in line of sight using dust texture
- Texture lockup only possible in fragment shader
- Assumes a Gaussian density profile in the vertical direction (pseudo 3D)

Off screen framebuffer

- Size 256 x 256
- One pixel for every vertex
- No blending

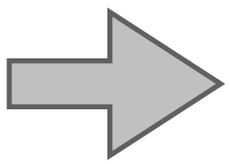
Render Step 11

256^2 vertices, pre-calculated opacity, point texture



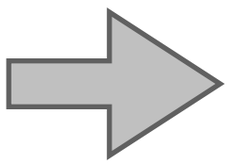
Vertex shader

- Transform positions of point according to viewing geometry



Fragment shader

- Two texture lookups:
- First, look up pre-calculated opacity
- Second, use point texture



On screen framebuffer

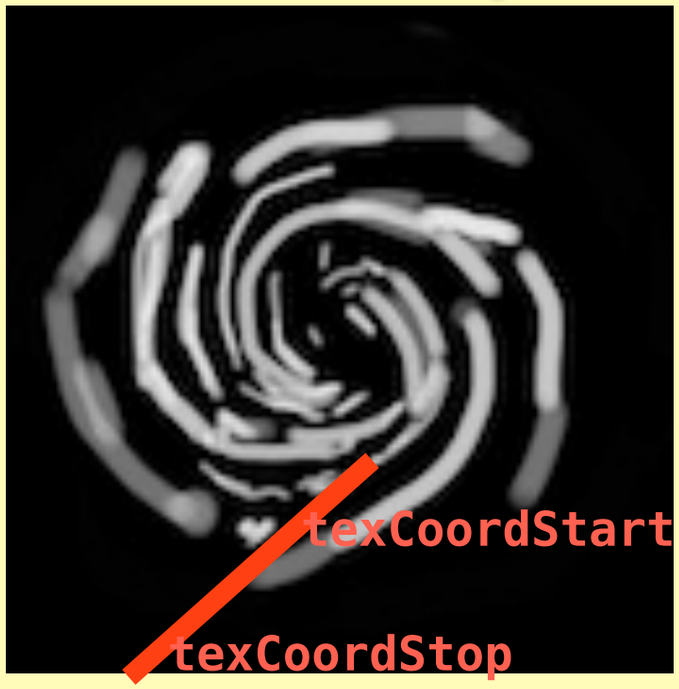
- Blending

Render Step 1

Vertex shader

```
//In
attribute vec4 Position;
attribute vec2 texCoordIn;

uniform mat4 modelViewMatrix;
uniform vec3 cameraPosition;

//Out
varying highp vec3 texCoordStart;
varying highp vec3 texCoordStop;
varying highp float lengthInDust;

void main(){
   vec4 modelViewPosition = modelViewMatrix * Position;

   gl_Position = vec4(texCoordIn.x,texCoordIn.y,0.,1.);
   gl_PointSize = 1.;

       ...

   texCoordStart =
   texCoordStop  =
}
```

Dust map

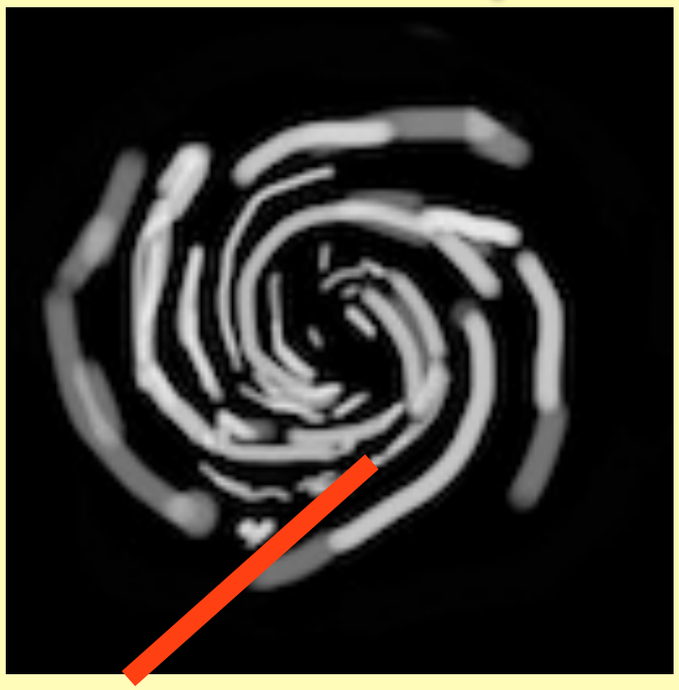


Render Step 1

Fragment shader

```
// In
uniform sampler2D SamplerDust;

varying highp vec3 texCoordStart;
varying highp vec3 texCoordStop;
varying highp float lengthInDust;

void main(){
   highp float transmission = 1.;
   highp float transmissionScaleFactor =
                  lengthInDust/milkyWayHeight/5.;

   for (int i=0;i<9;i++){
    highp float frac = float(i)/10.;
    highp vec3 texCoord = texCoordStart*frac+texCoordStop*(1.-frac);

    highp float z = (texCoord.z-0.5)*milkyWayScale/milkyWayHeight*4.;
    highp float dustopacity =
             texture2D(SamplerDust, texCoord.xy).a*exp(-z*z);

    transmission *= exp(-dustopacity*transmissionScaleFactor);
   }

   gl_FragColor = vec4(1.,1.,1.,-0.1+1.2*transmission);
}
```

Dust map



Render Step 1

Calling code

```
// Bind off screen framebuffer
glBindFramebuffer(GL_FRAMEBUFFER, frameBufferID);
glBindRenderbuffer(GL_RENDERBUFFER, colorRenderBuffer);
glViewport(0, 0, 256, 256);
glClearColor(0, 0, 0, 1.f);
glClear(GL_COLOR_BUFFER_BIT);


SimpleShader* pointShader = [ShaderManager
    getSimpleShaderWithName:@"PointsMilkyWayToTexture"];
glUseProgram(pointShader.program);
glDisable(GL_BLEND);

// Setup dust texture
[ExoplanetSharegroup bindTexture:@"milkywaydust.png"];
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

// Provide matricies and uniforms
glUniformMatrix4fv([pointShader getUniform:@"modelViewMatrix"],
     1, 0, _modelViewMatrix.m);
GLKVector3 cameraPosition =
```

Render Step II

Vertex shader

```
//In
attribute vec4 Position;
attribute float PointSize;
attribute vec4 Color;
attribute vec2 texCoordIn;

uniform highp float pointSizePreMultiply;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

//Out
varying highp vec4 colorVarying;
varying highp vec2 texCoordOut;

void main(){
   vec4 modelViewPosition = modelViewMatrix * Position;
   gl_Position  = projectionMatrix * modelViewPosition;
   gl_PointSize = max(1.,PointSize*pointSizePreMultiply);
   colorVarying = Color;
   texCoordOut  = texCoordIn/2.+0.5;
}
```

Render Step II

Fragment shader

```
// In
uniform sampler2D Sampler;
uniform sampler2D SamplerDust;

varying highp vec4 colorVarying;
varying highp vec2 texCoordOut;

void main(){
  highp float transmission = texture2D(SamplerDust,texCoordOut).a;
  gl_FragColor = vec4(1.,1.,1.,
       texture2D(Sampler, gl_PointCoord).a*transmission)
       *colorVarying;
}
```

Render Step 11

Calling code

```
SimpleShader* pointShader = [ShaderManager getSimpleShaderWithName:@"PointsMilkyWay"];
glUseProgram(pointShader.program);
glEnable (GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

// Setup textures
glUniform1i([pointShader getUniform:@"SamplerDust"], 1);
glUniform1i([pointShader getUniform:@"Sampler"], 0);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, colorRenderBufferTextureID);
glActiveTexture(GL_TEXTURE0);
[ExoplanetSharegroup bindTexture:@"newblob.png"];


// Provide matricies and uniforms
glUniformMatrix4fv([pointShader getUniform:@"modelViewMatrix"], 1, 0, _modelViewMatrix.m);
glUniformMatrix4fv([pointShader getUniform:@"projectionMatrix"], 1, 0, _projectionMatrix.m);
glUniform1f([pointShader getUniform:@"UserScale"], _userScale);
float pointSize = MIN(16./_userScale,8./sqrtf(_userScale));
if (isRetina)   pointSize *= 2.;
if (isIpad)     pointSize *= 2.;
glUniform1f([pointShader getUniform:@"pointSizePreMultiply"], pointSize);

glBindBuffer(GL_ARRAY_BUFFER, vertexBuffer);
glVertexAttribPointer([pointShader getAttribute:@"Position"],  3,
    GL_FLOAT, GL_FALSE, sizeof(float)*10, 0);
glVertexAttribPointer([pointShader getAttribute:@"PointSize"],  1,
    GL_FLOAT, GL_FALSE, sizeof(float)*10, BUFFER_OFFSET(sizeof(float)*3));
```

Take home messages

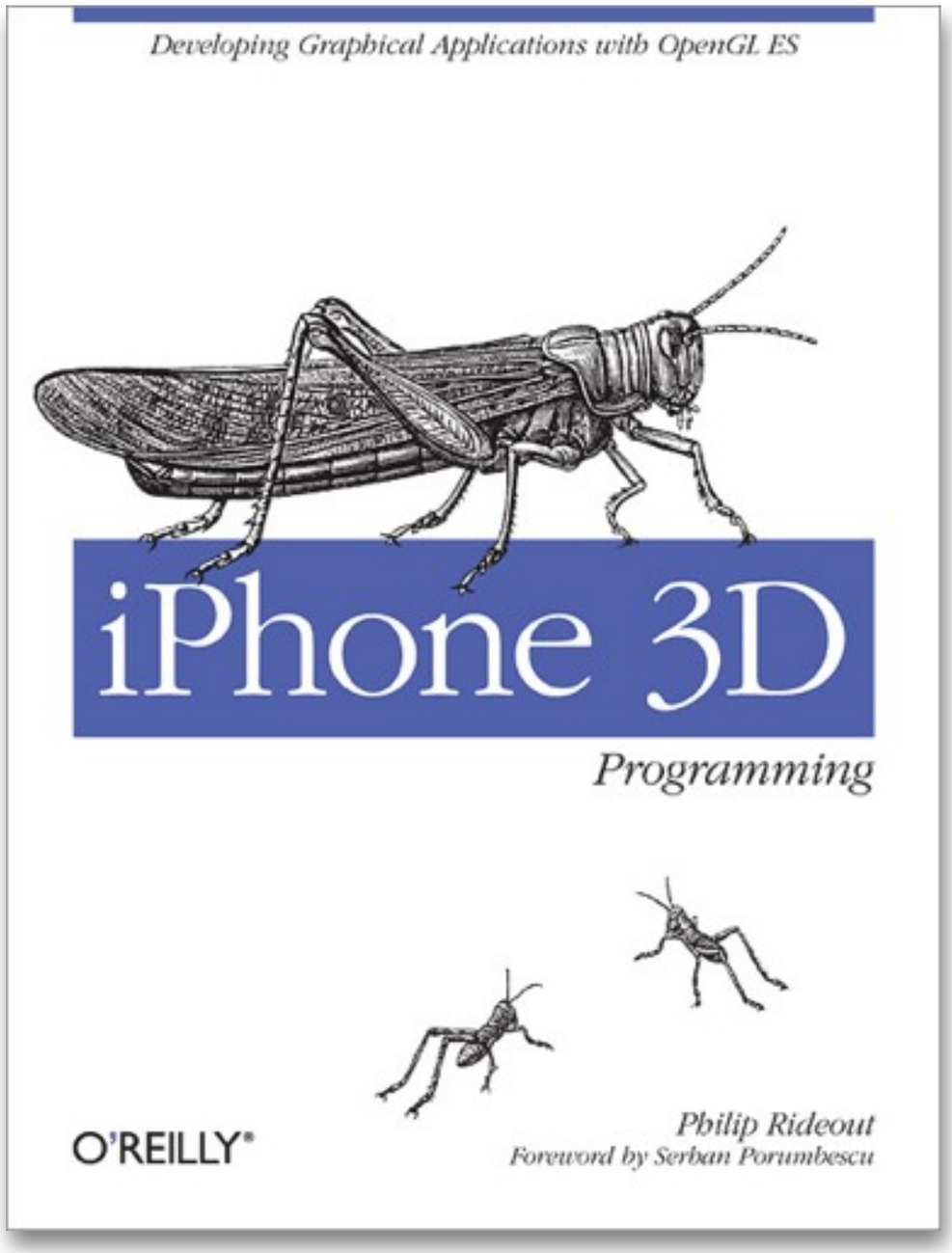
Take home messages

Start playing with OpenGL

- Use the simplest working example and expand it.
- Don't worry too much about how to set up the view, etc.

Think out of the box

- You don't have to use OpenGL for what it was built for.
- Use it to do a large parallel calculation on the GPU.



Thank you!

Source code: github.com/hannorein/OpenGLMilkyWay
E-mail: hanno@hanno-rein.de
Book: iPhone 3D Programming
by Philip Rideout, O'Reilly Media