BLUECADET

Shader ~~Lunch~~ Drink & Learn

Ben Bojko – February 22, 2017

BLUECADET

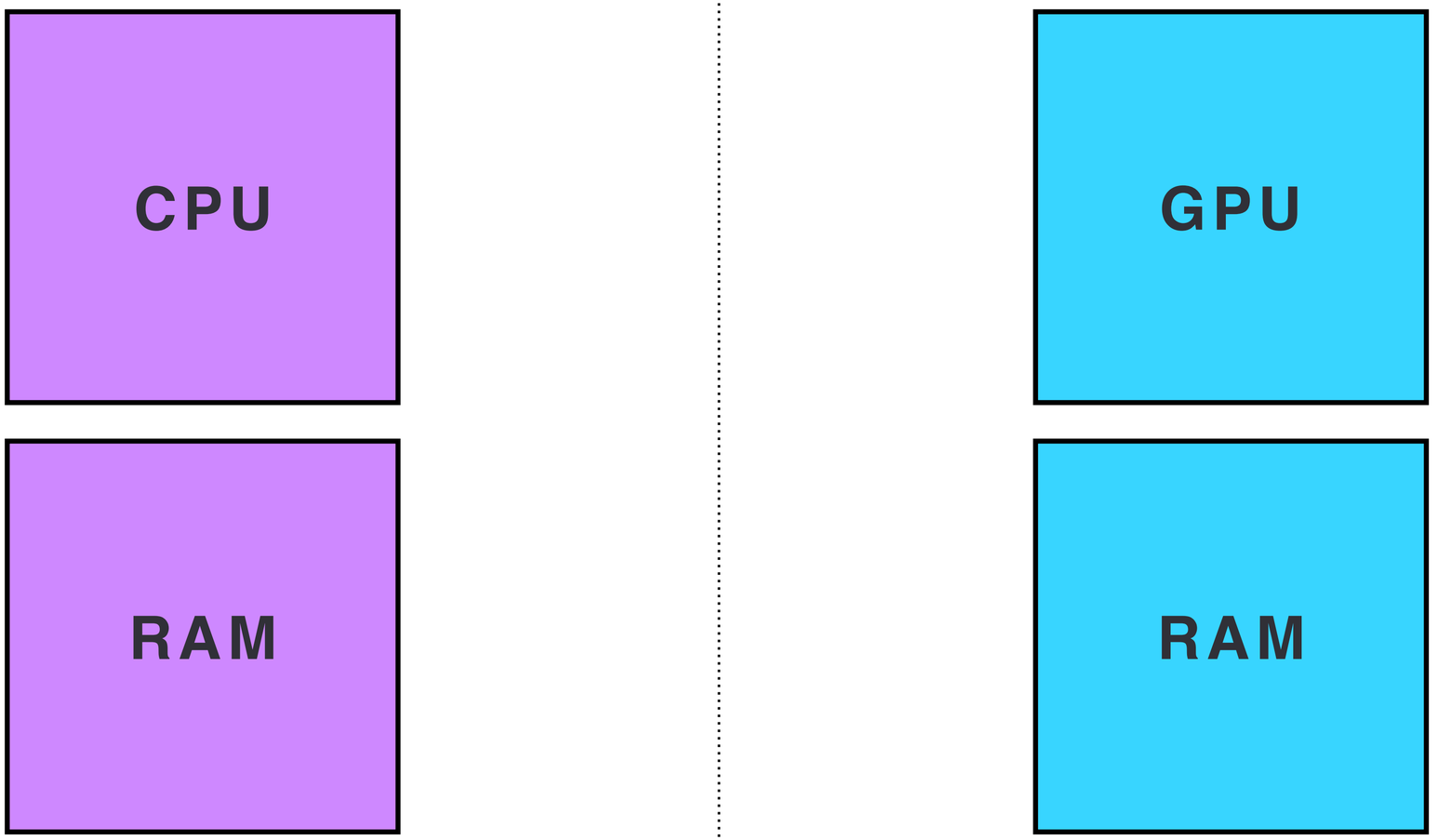
AGENDA

1. Intro
    1. When To Use Shaders
    2. OpenGL Basics
    3. Shader Basics
2. Examples

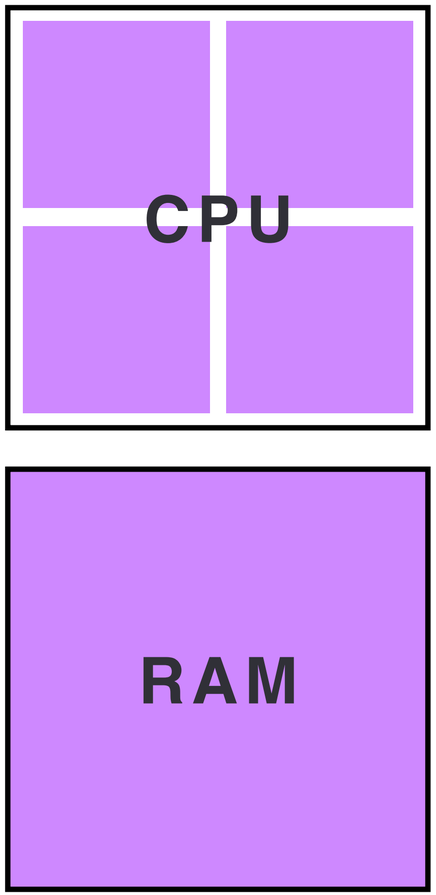
INTRO

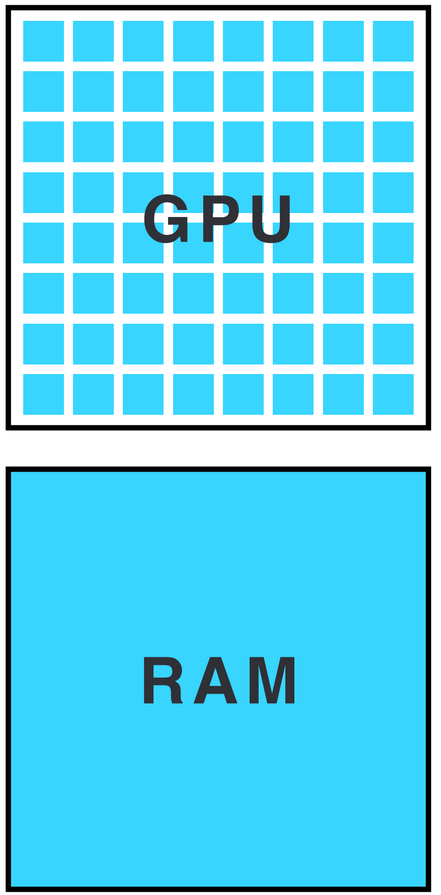
What are Shaders?

CPU vs GPU



CPU vs GPU





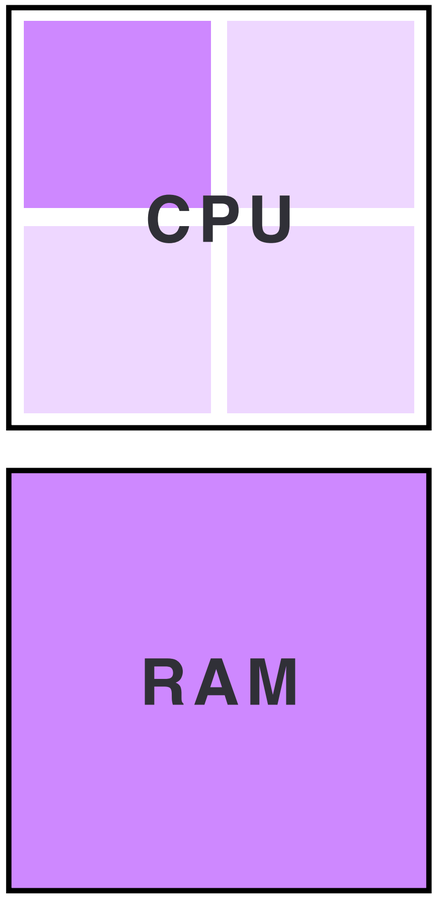
WHEN TO USE SHADERS

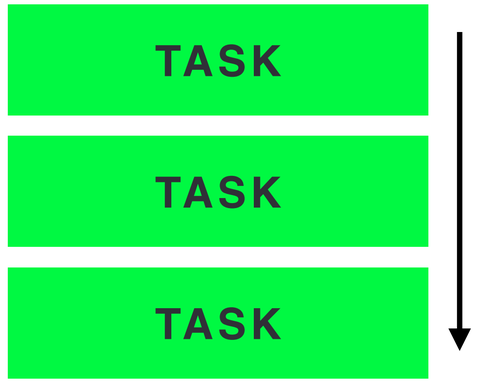
CPU vs GPU

CPU

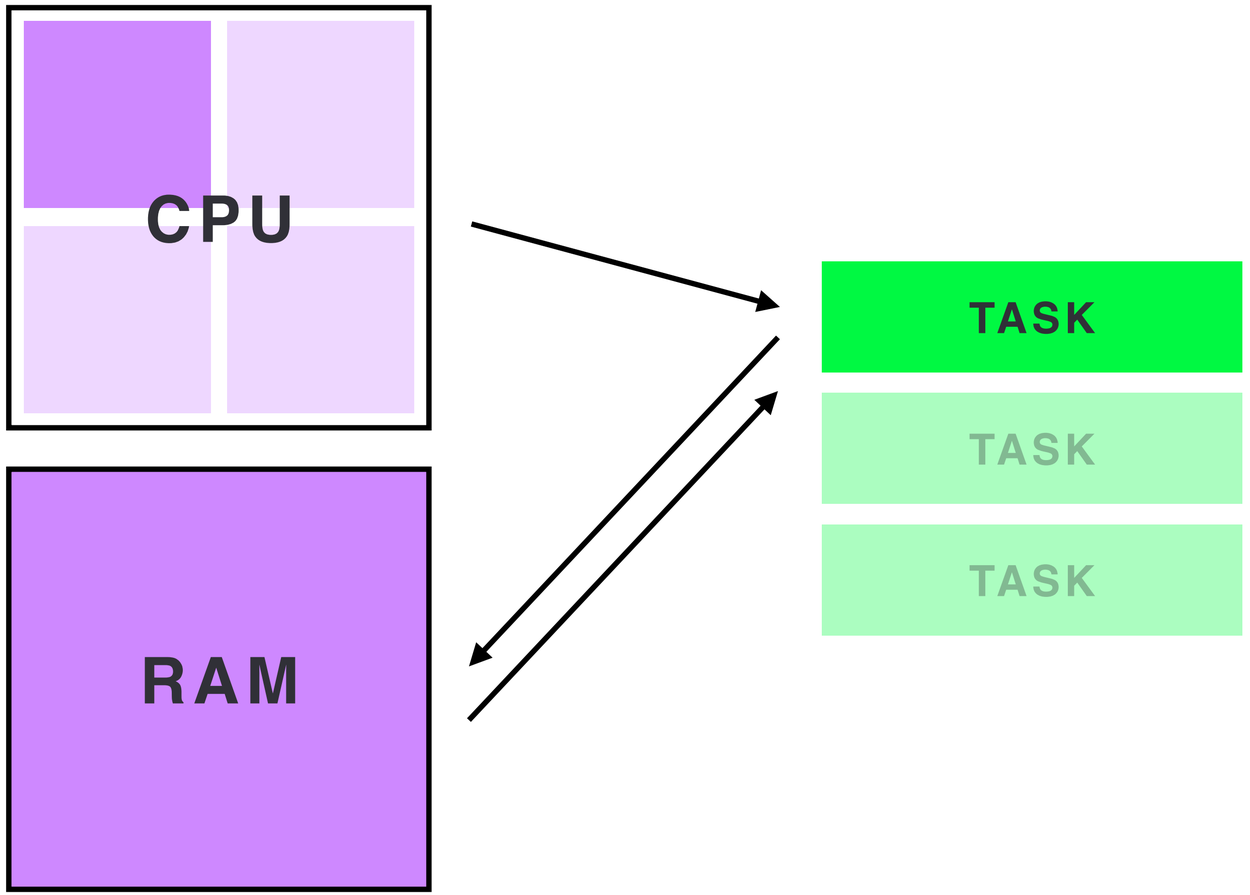
GPU

CPU Execution

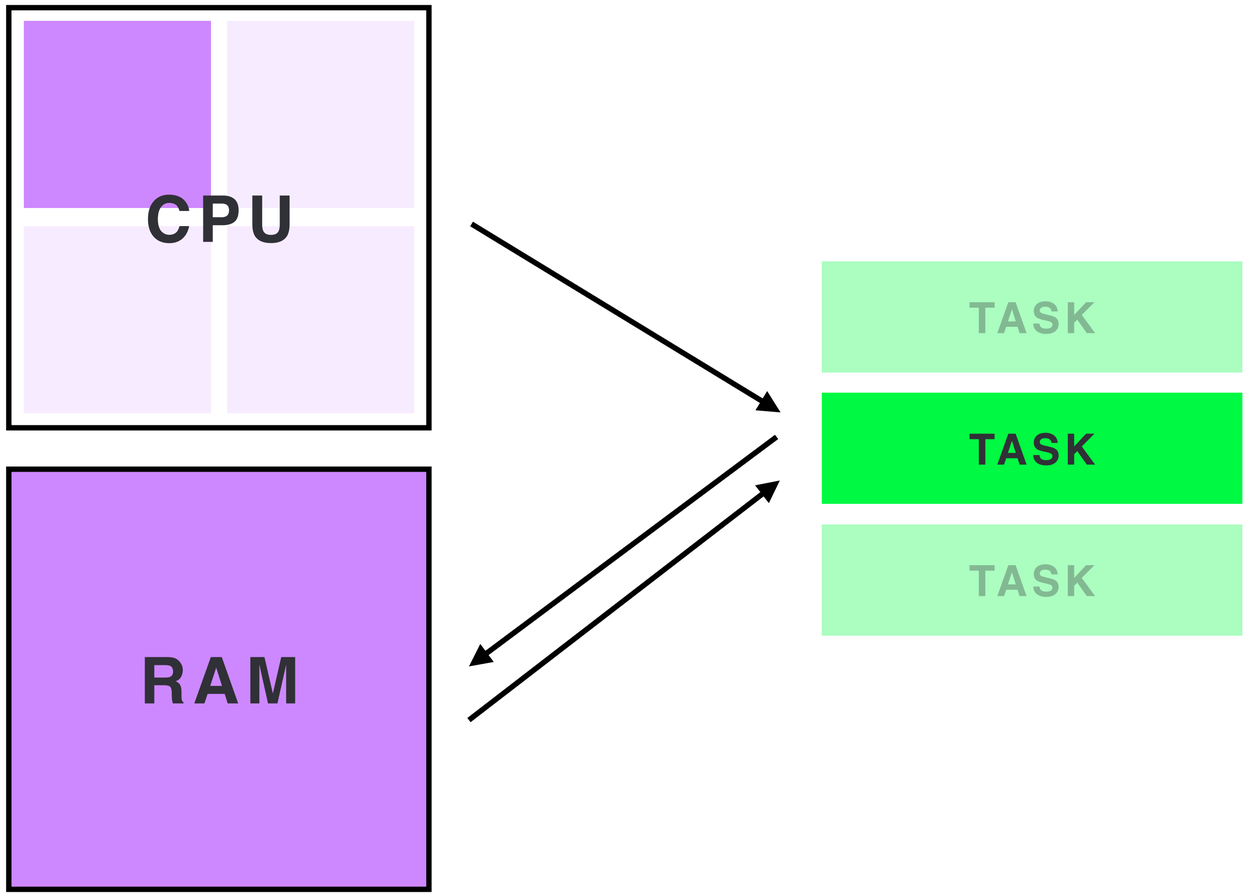




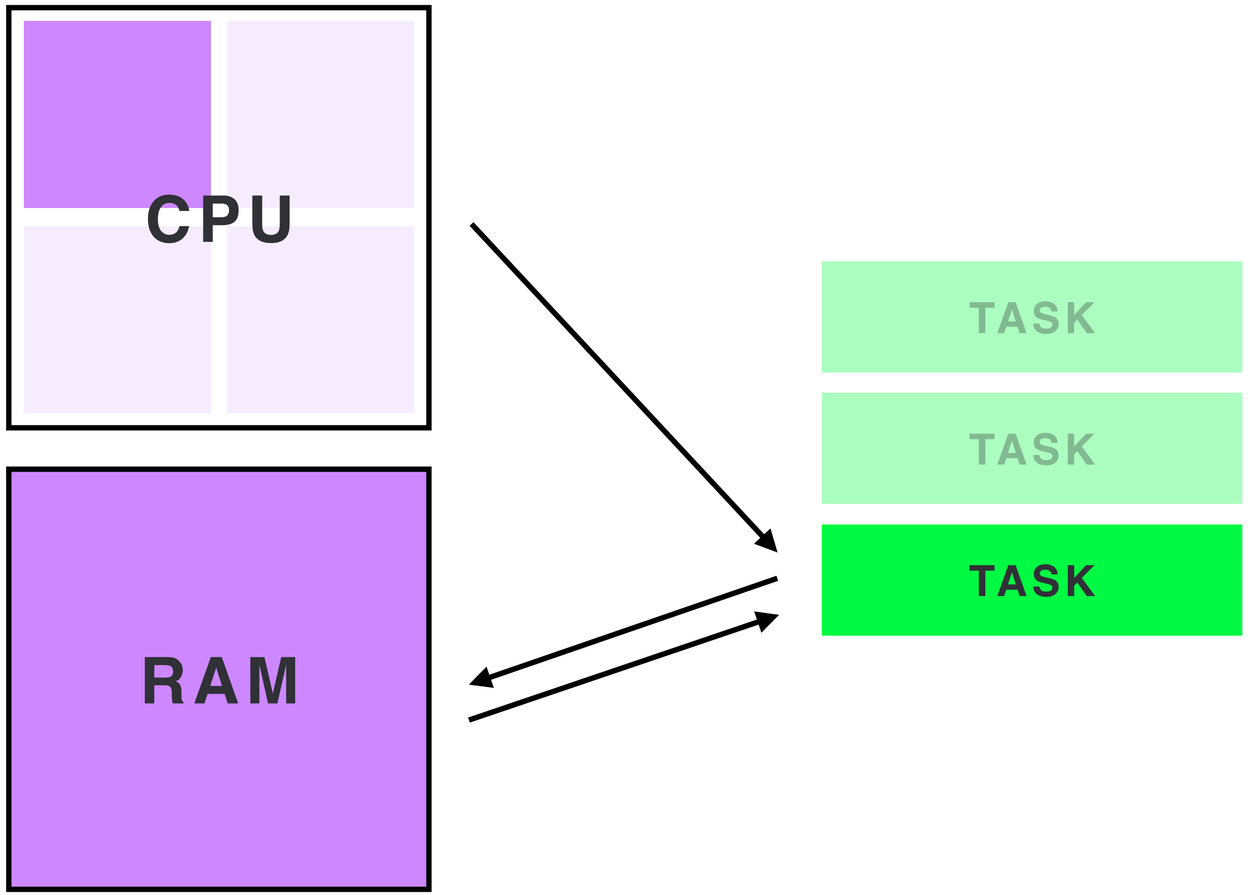
CPU Execution



CPU Execution

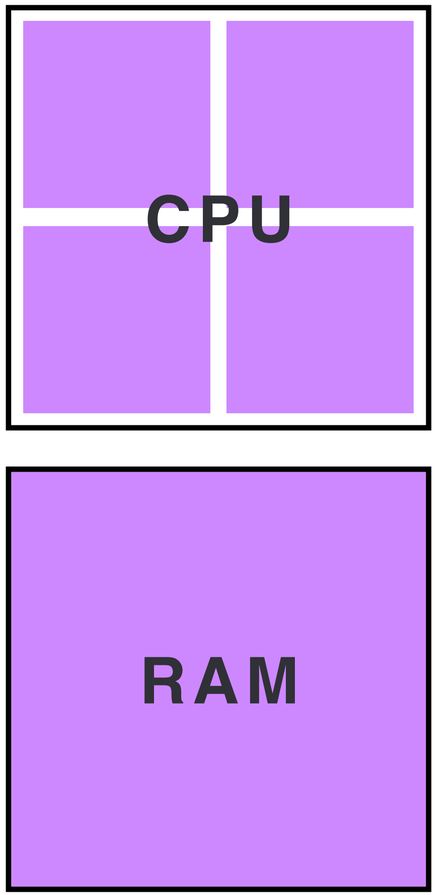


CPU Execution

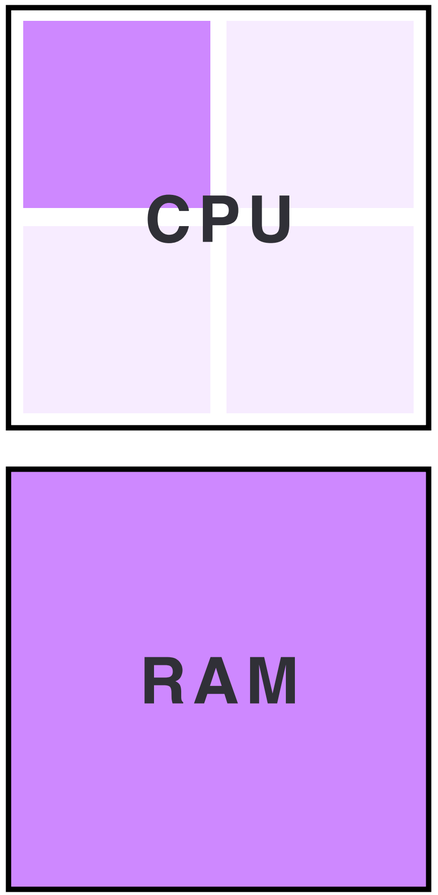


WHEN TO USE SHADERS

CPU Multithreading

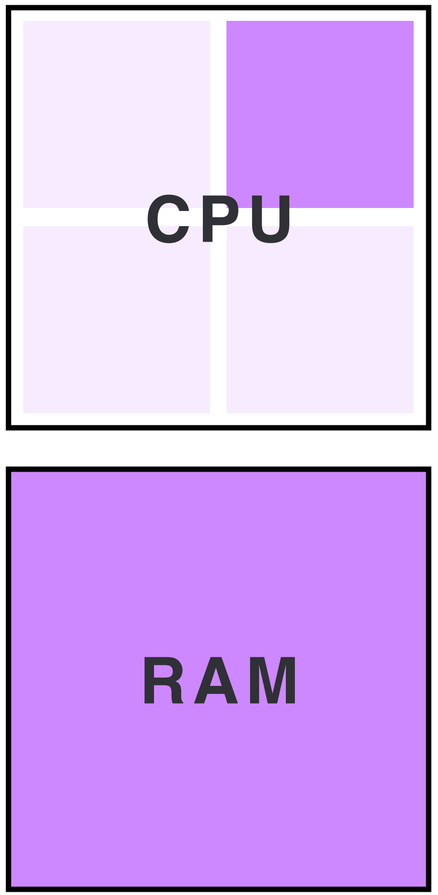


CPU Multithreading

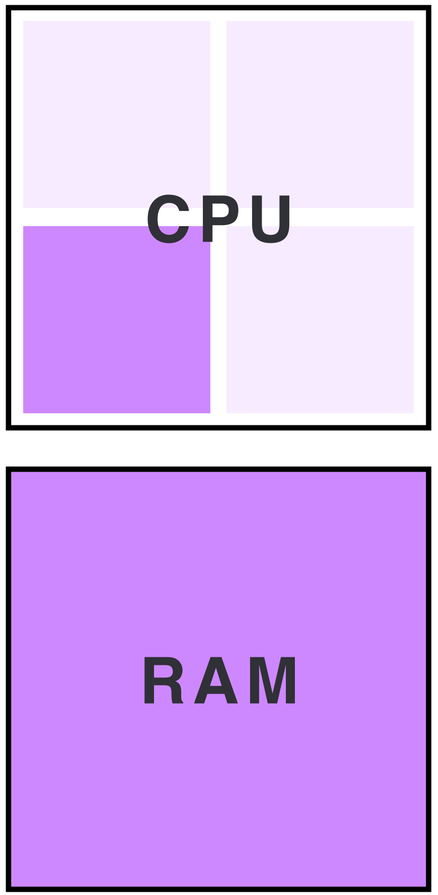


THREAD 1	THREAD 2	THREAD 3	THREAD 4
TASK	TASK	TASK	TASK
TASK	TASK	TASK	TASK
TASK	TASK	TASK	TASK

CPU Multithreading

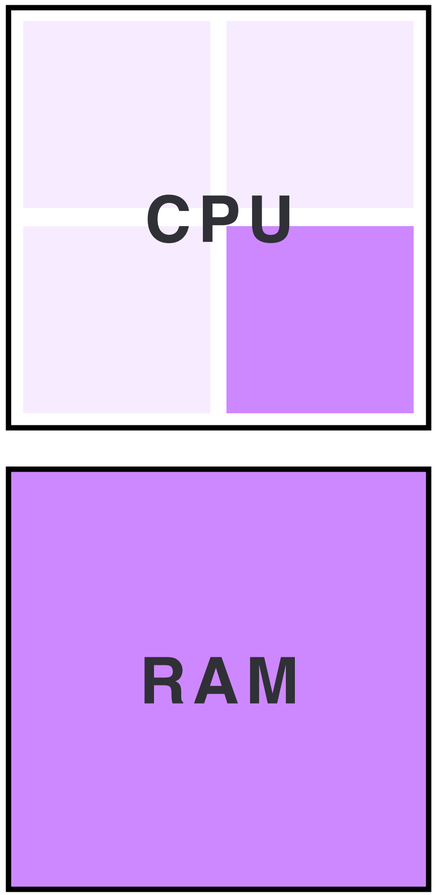


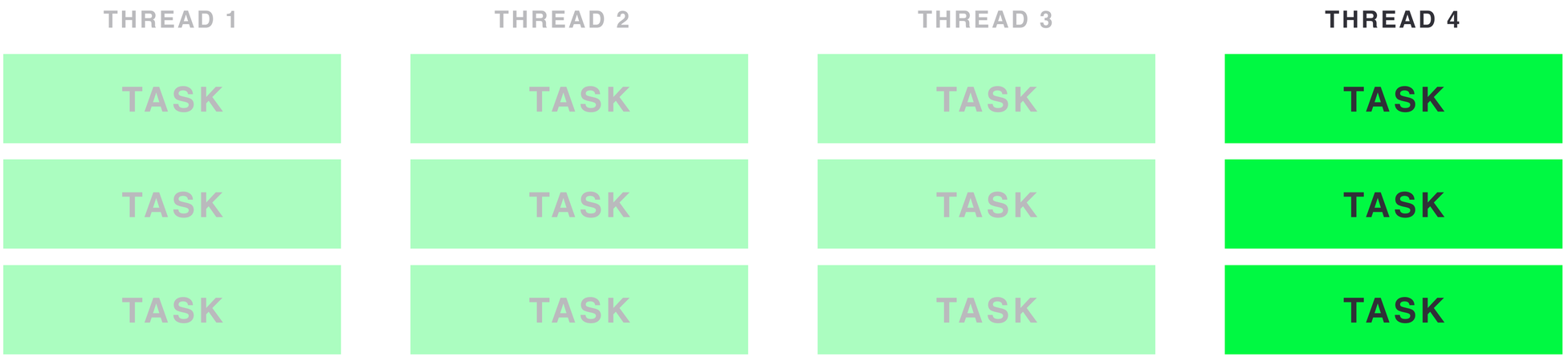
CPU Multithreading



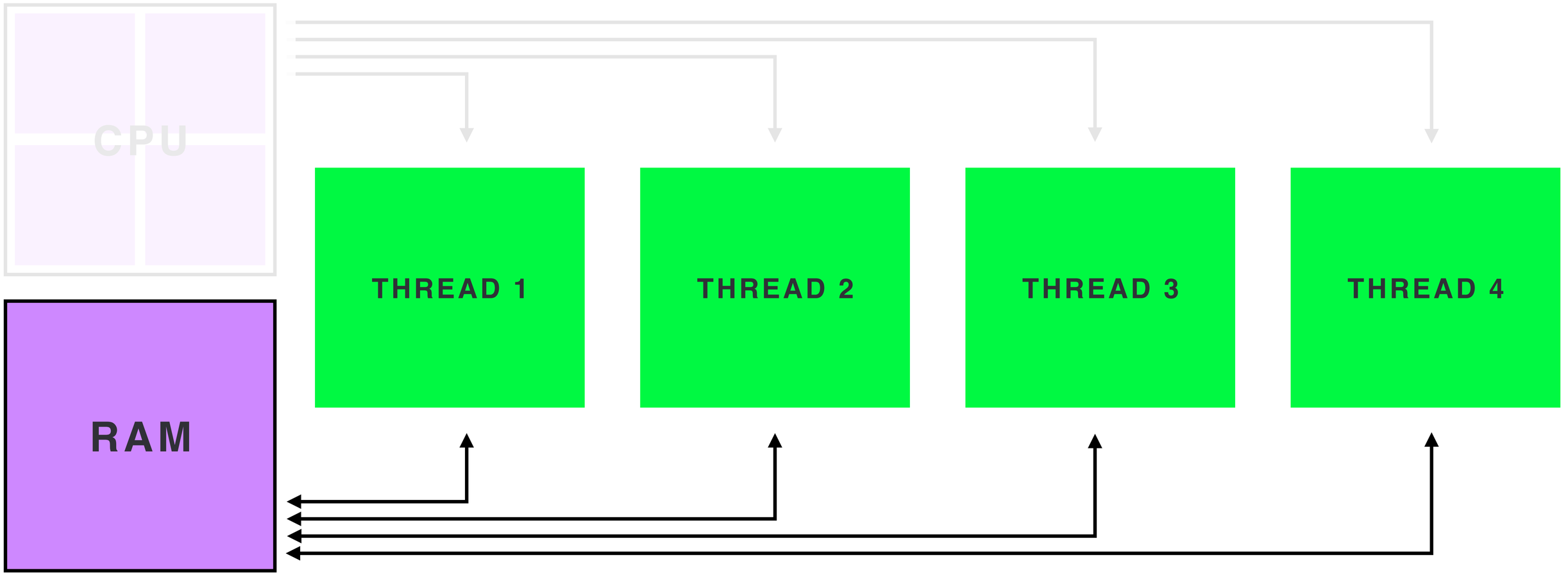
THREAD 1	THREAD 2	THREAD 3	THREAD 4
TASK	TASK	TASK	TASK
TASK	TASK	TASK	TASK
TASK	TASK	TASK	TASK

CPU Multithreading

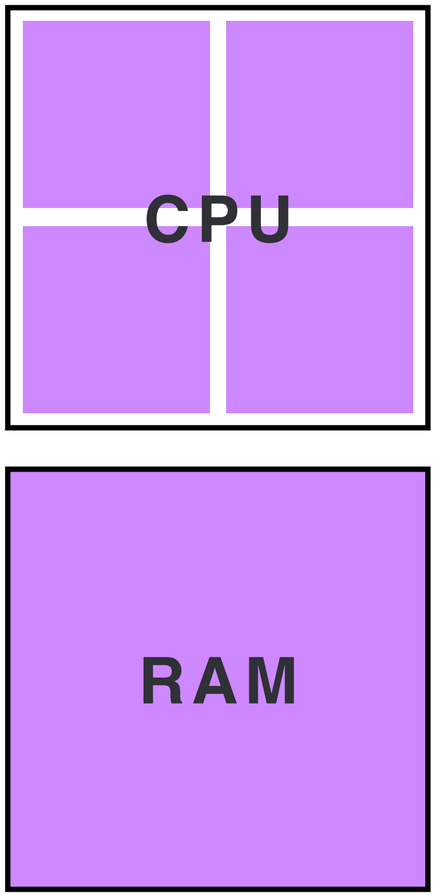


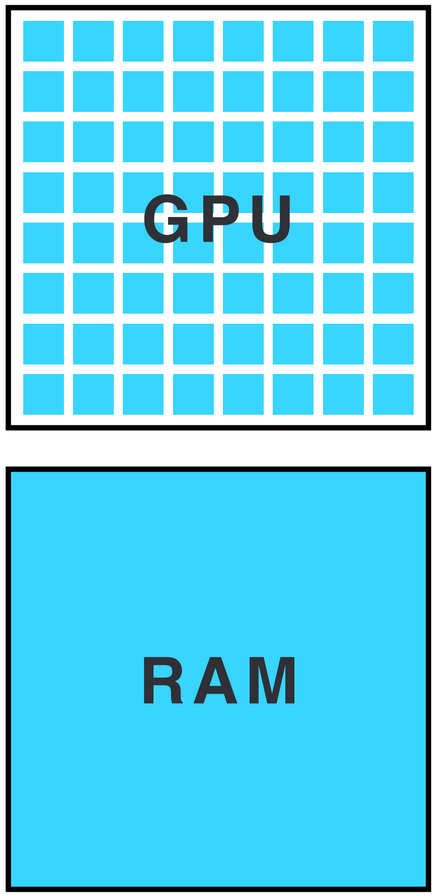


CPU Multithreading – Shared Memory

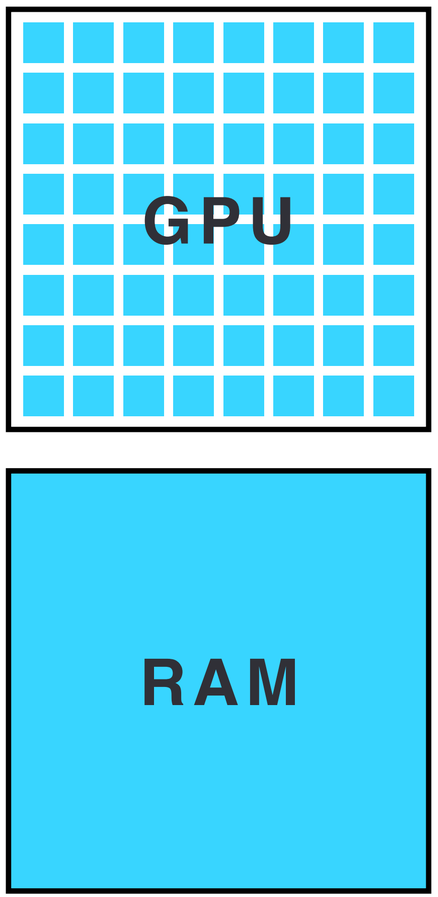


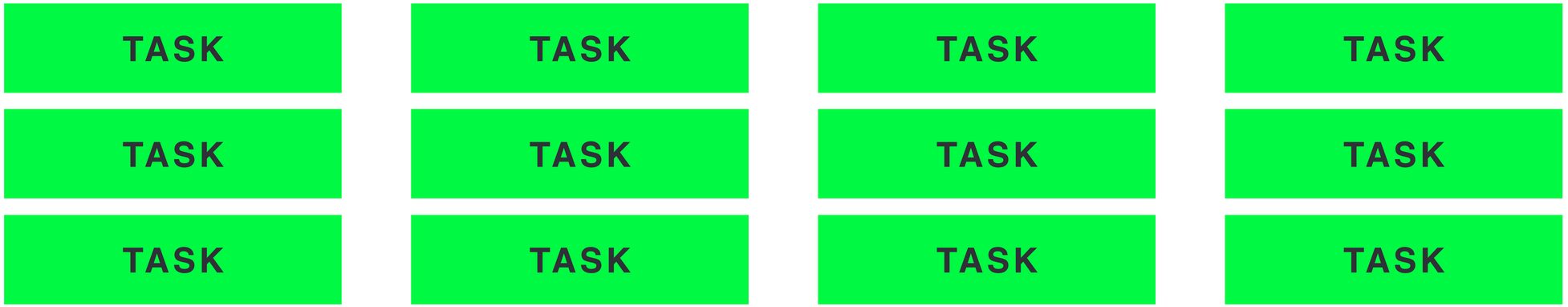
CPU vs GPU



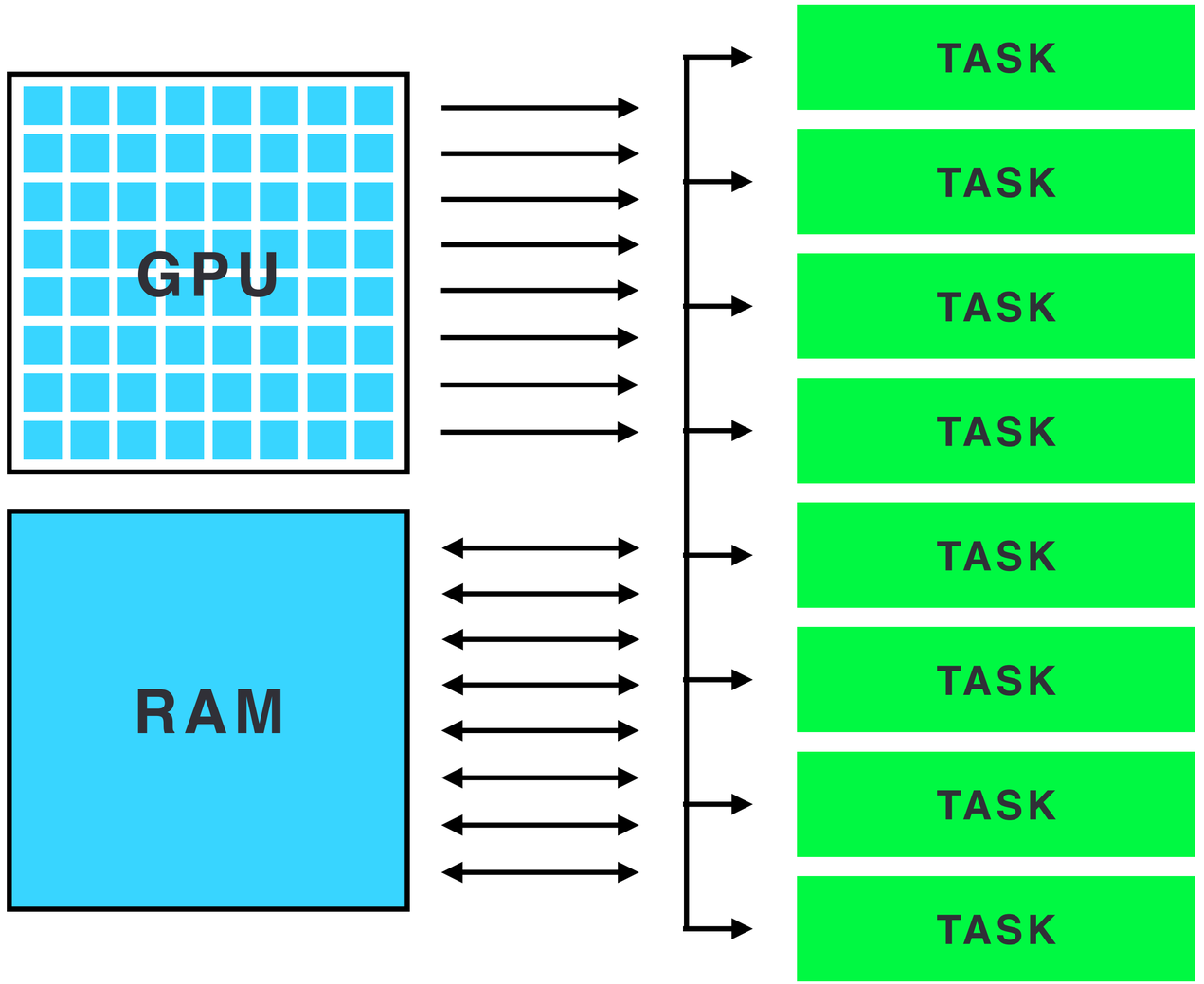


GPU Execution





GPU Execution

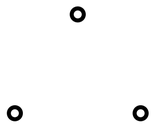


INTRO

OpenGL Basics

OPENGL BASICS

Graphics Pipeline



VERTEX DATA

TRANSFORMATION

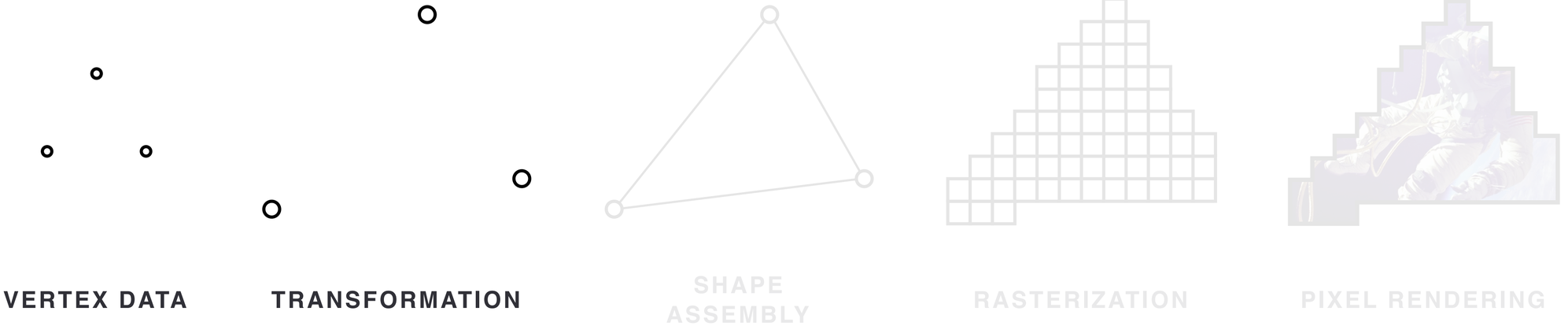
SHAPE ASSEMBLY

RASTERIZATION

PIXEL RENDERING

OPENGL BASICS

Graphics Pipeline



VERTEX DATA

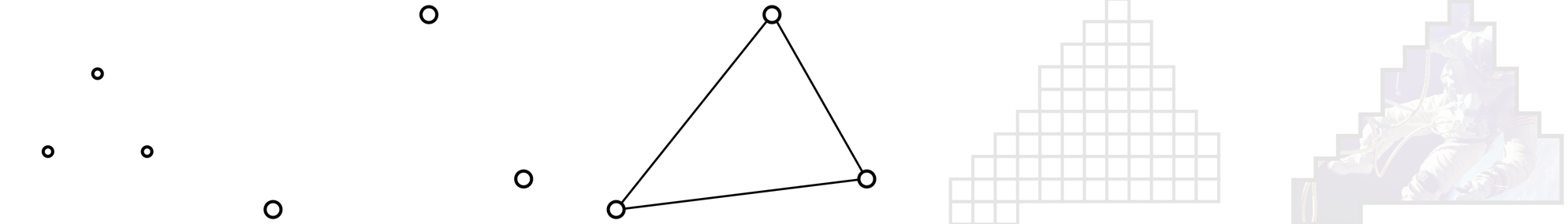
TRANSFORMATION

SHAPE ASSEMBLY

RASTERIZATION

PIXEL RENDERING

Graphics Pipeline



VERTEX DATA

TRANSFORMATION

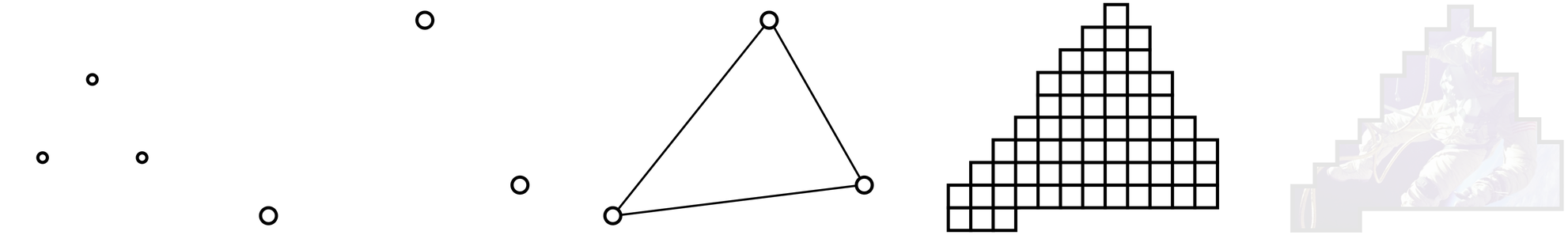
SHAPE ASSEMBLY

RASTERIZATION

PIXEL RENDERING

OPENGL BASICS

Graphics Pipeline



VERTEX DATA

TRANSFORMATION

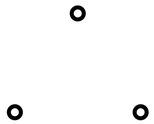
SHAPE ASSEMBLY

RASTERIZATION

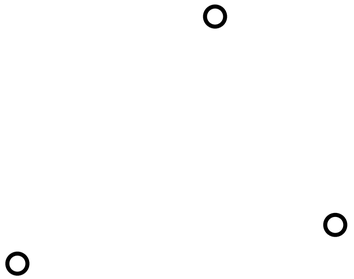
PIXEL RENDERING

OPENGL BASICS

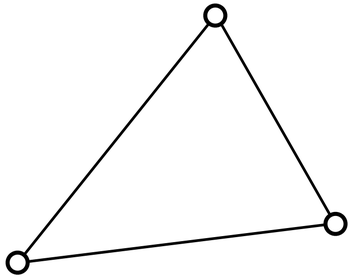
Graphics Pipeline



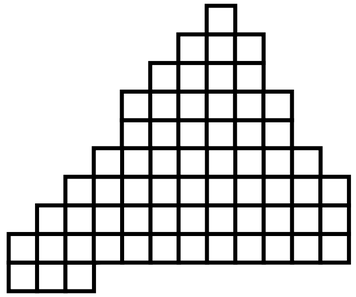
VERTEX DATA



TRANSFORMATION



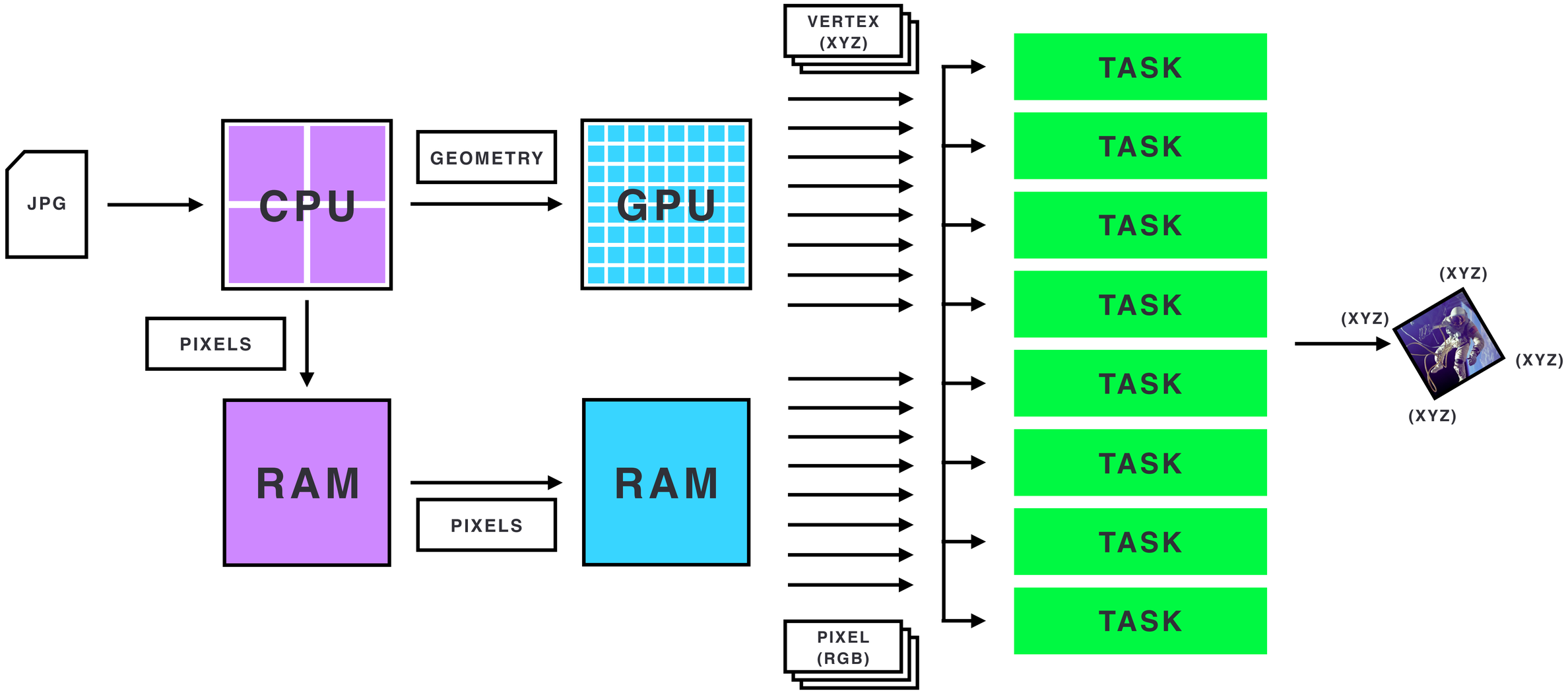
SHAPE
ASSEMBLY



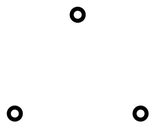
RASTERIZATION

PIXEL RENDERING

Example: Rendering an Image



Batches



VERTEX DATA

+

...

TEXTURES

TRANSFORMS

TEXTURES

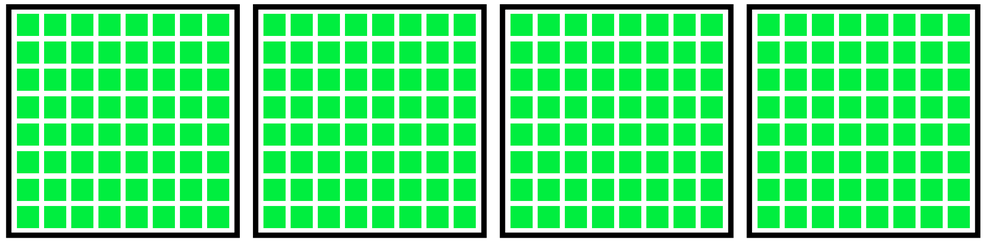
RENDER STATE

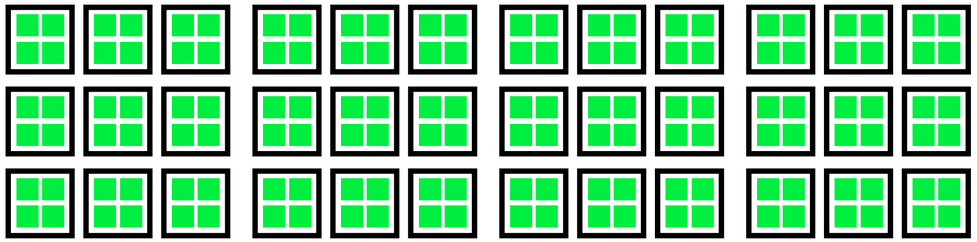
Batches

CPU TO GPU BOTTLENECK

Better to draw **10 batches** with **10 million objects** each than to draw **10 million batches** with **10 objects** each.

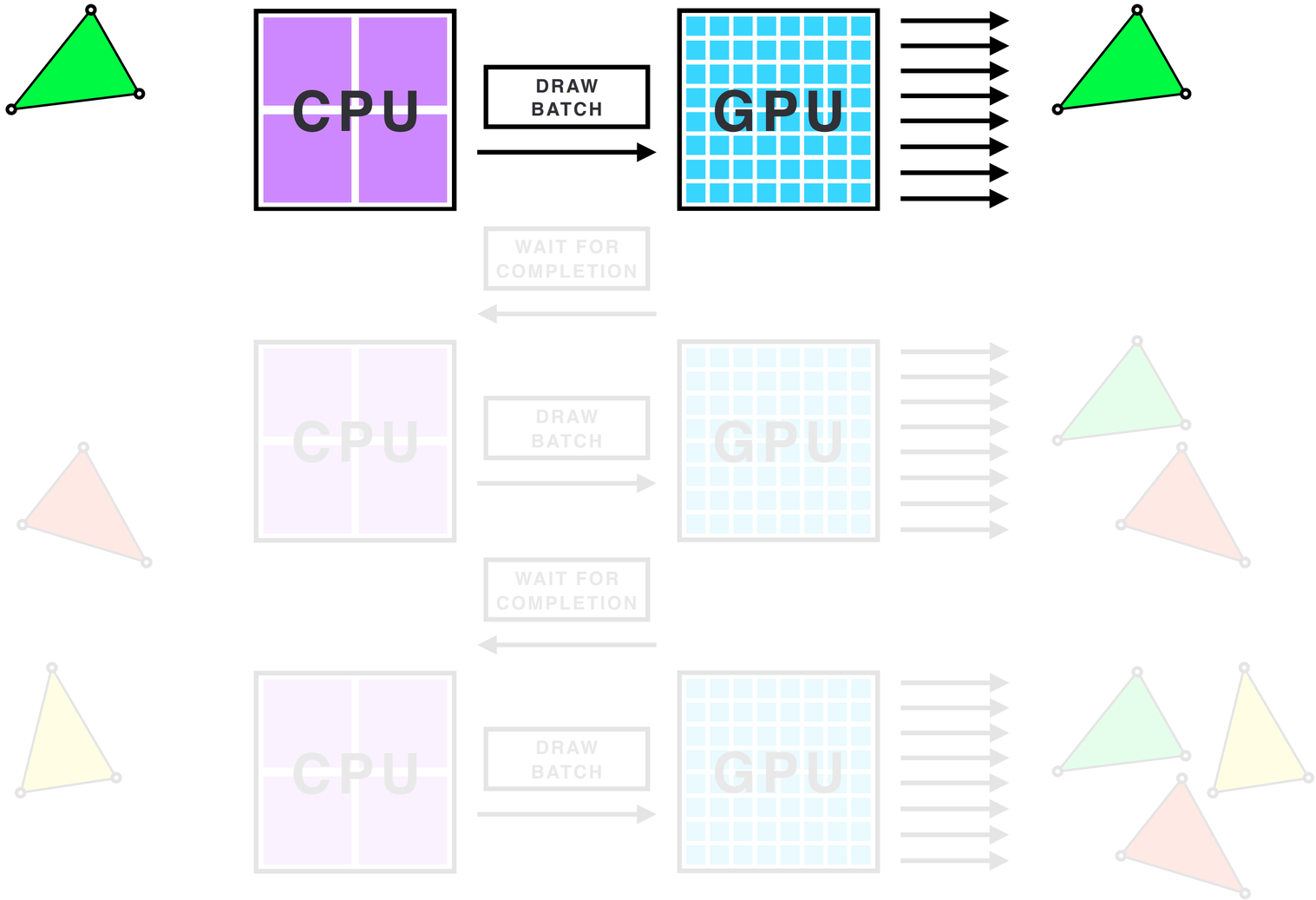
FEW BATCHES WITH MANY OBJECTS



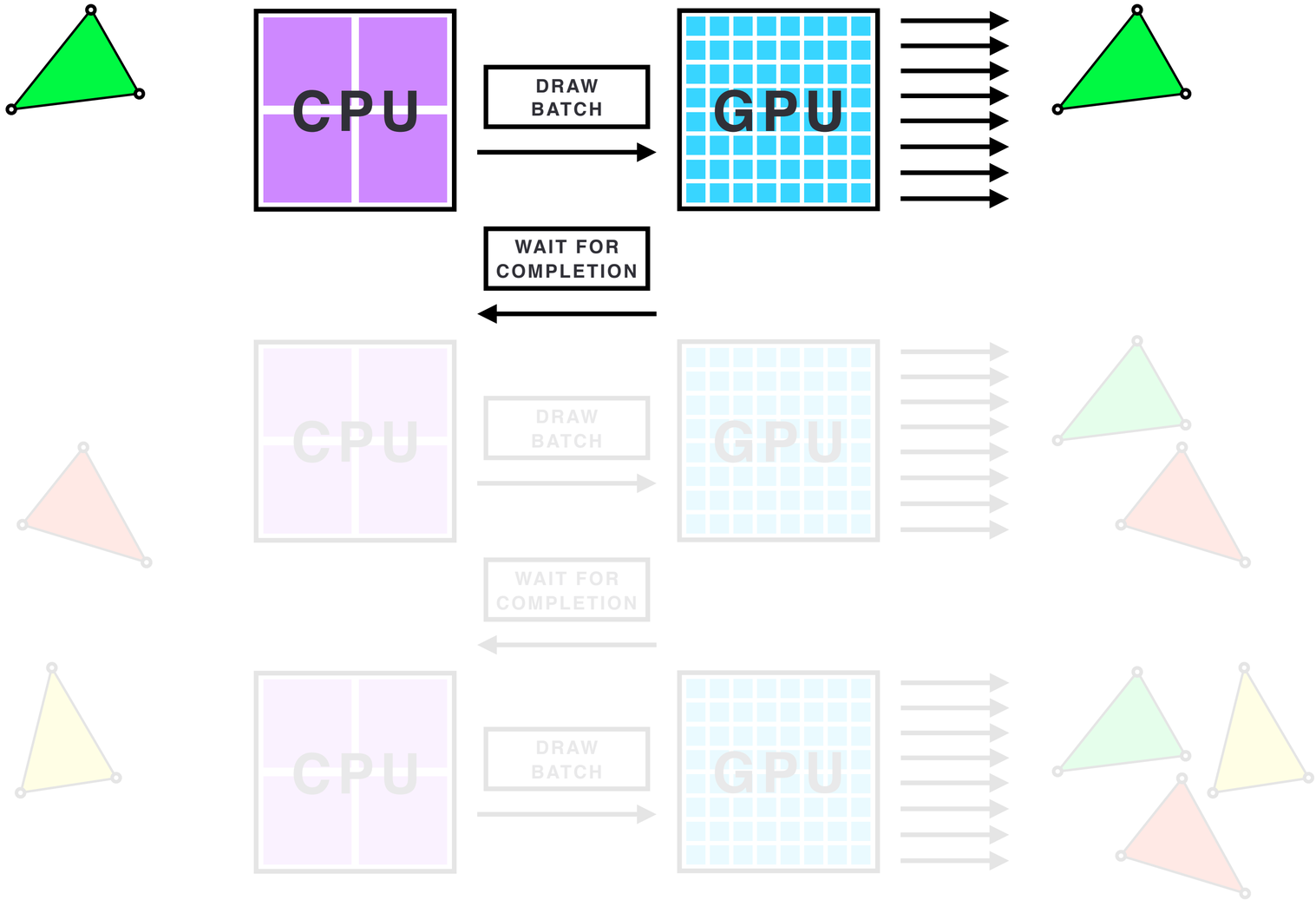


MANY BATCHES WITH FEW OBJECTS

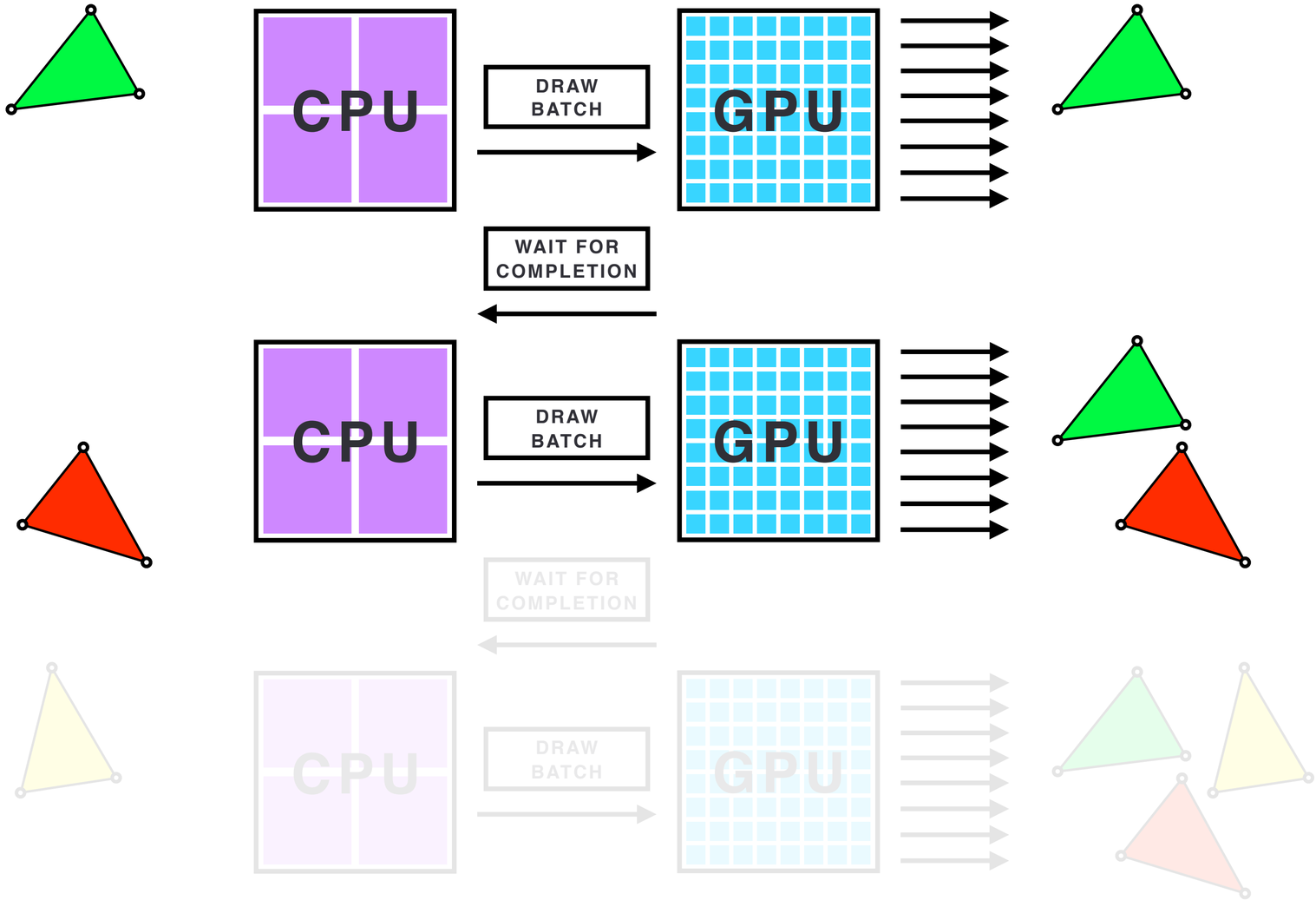
Example: CPU to GPU Bottleneck



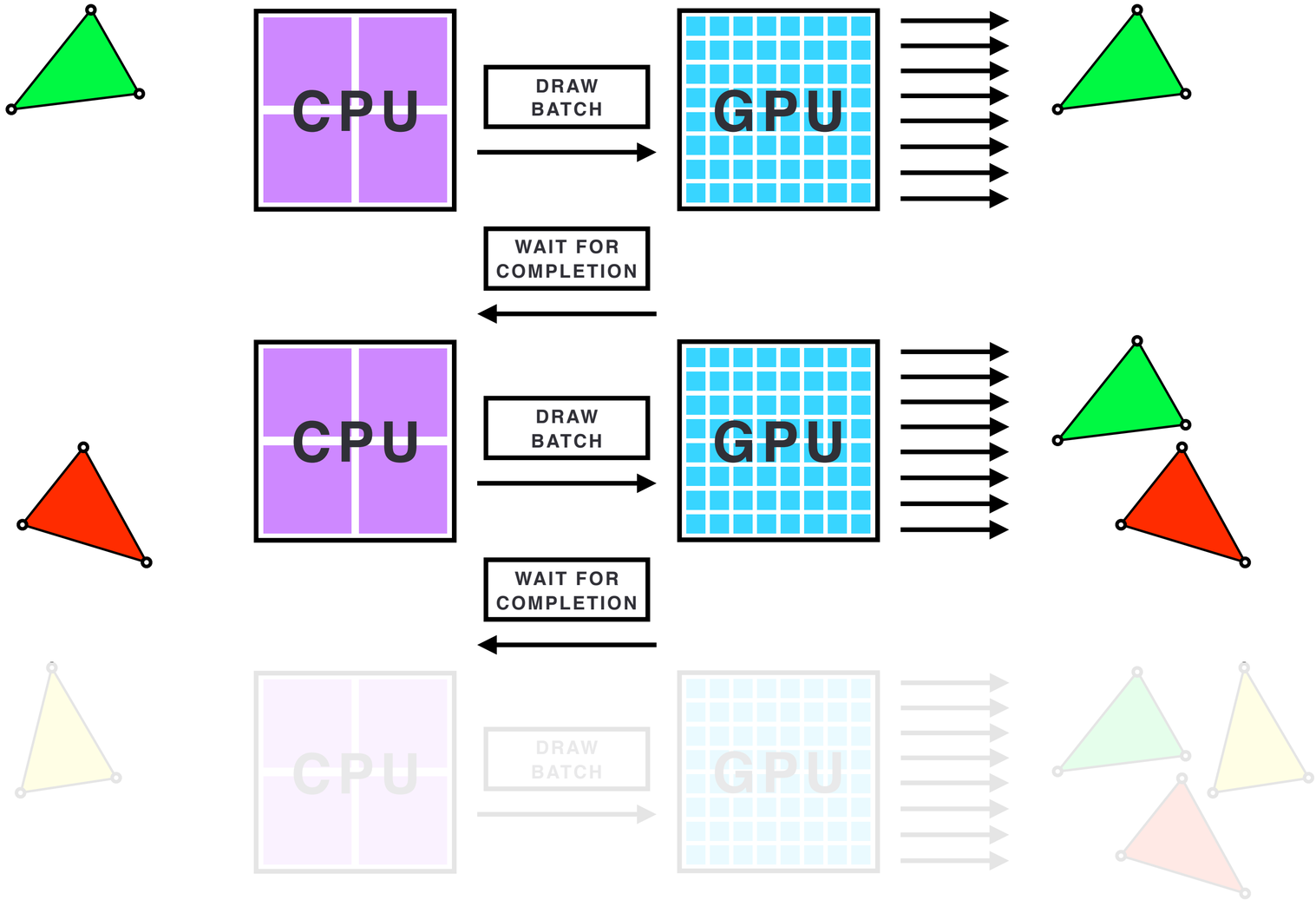
Example: CPU to GPU Bottleneck



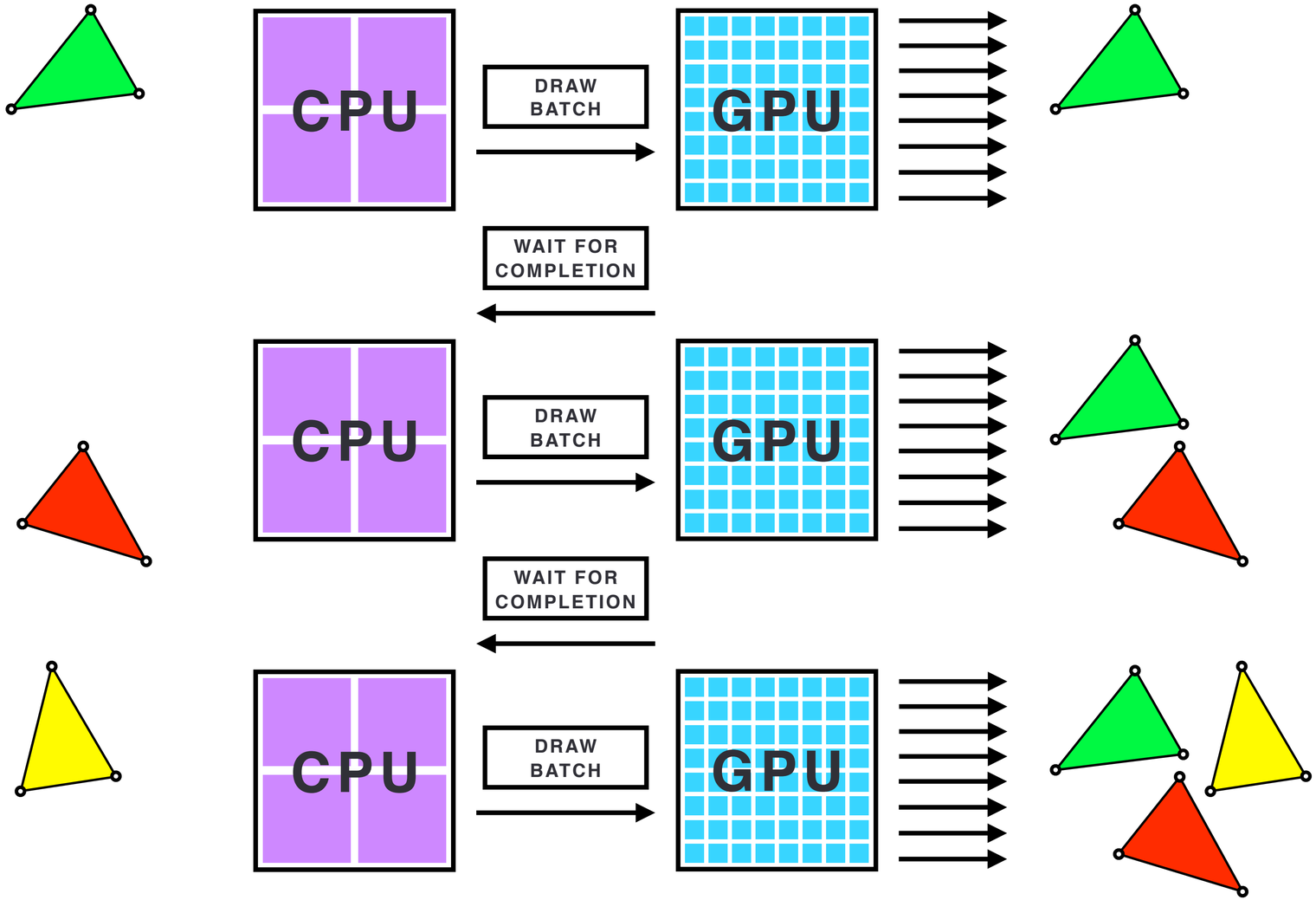
Example: CPU to GPU Bottleneck



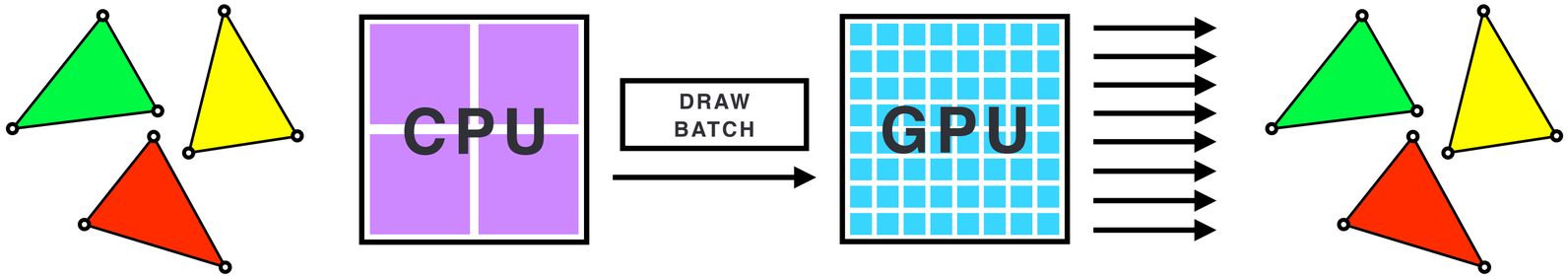
Example: CPU to GPU Bottleneck



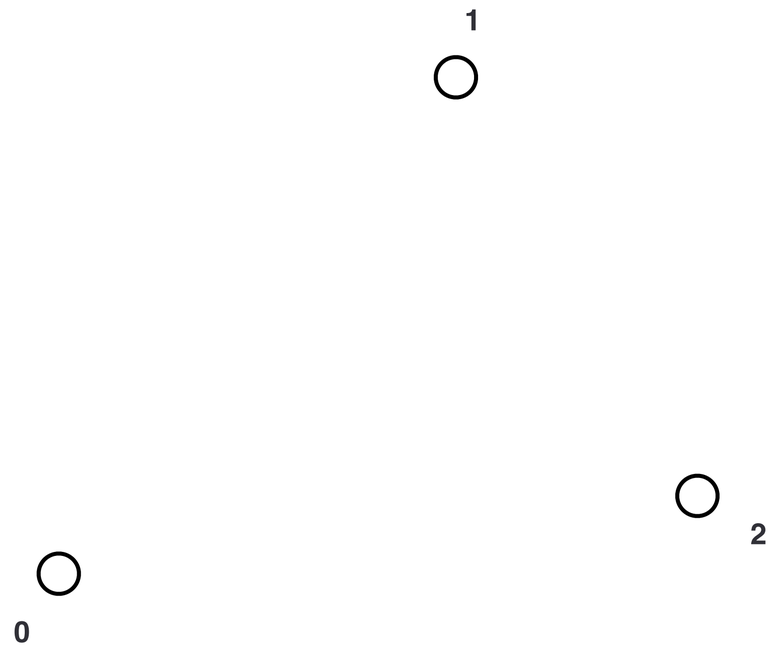
Example: CPU to GPU Bottleneck



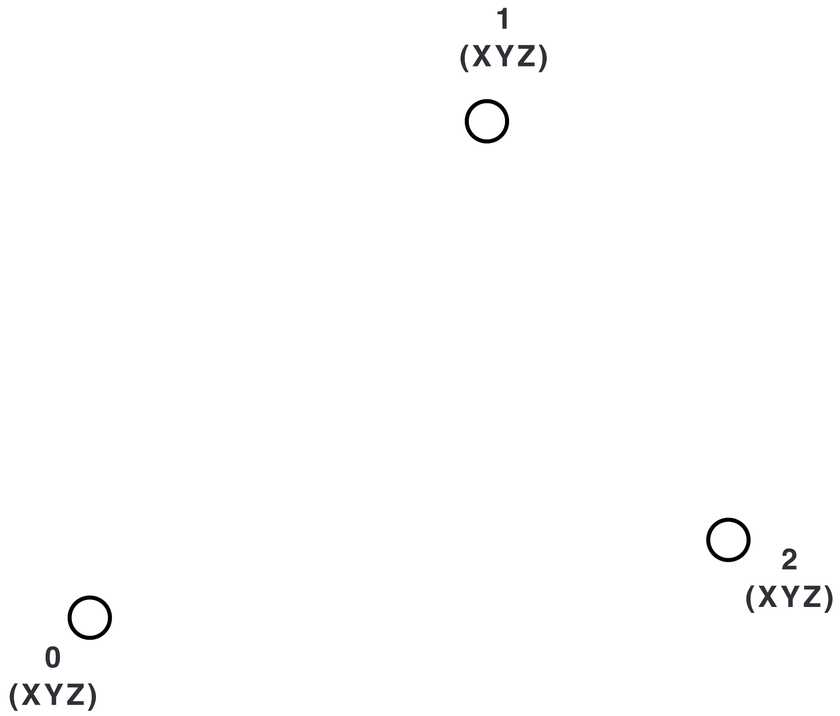
Example: CPU to GPU Bottleneck



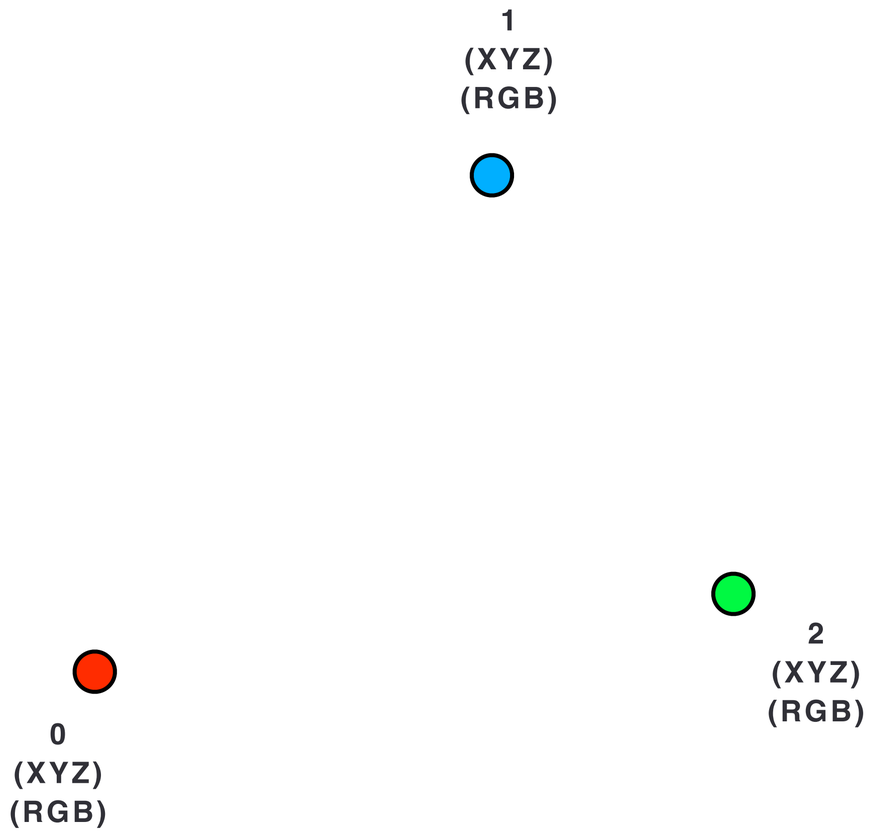
Vertices



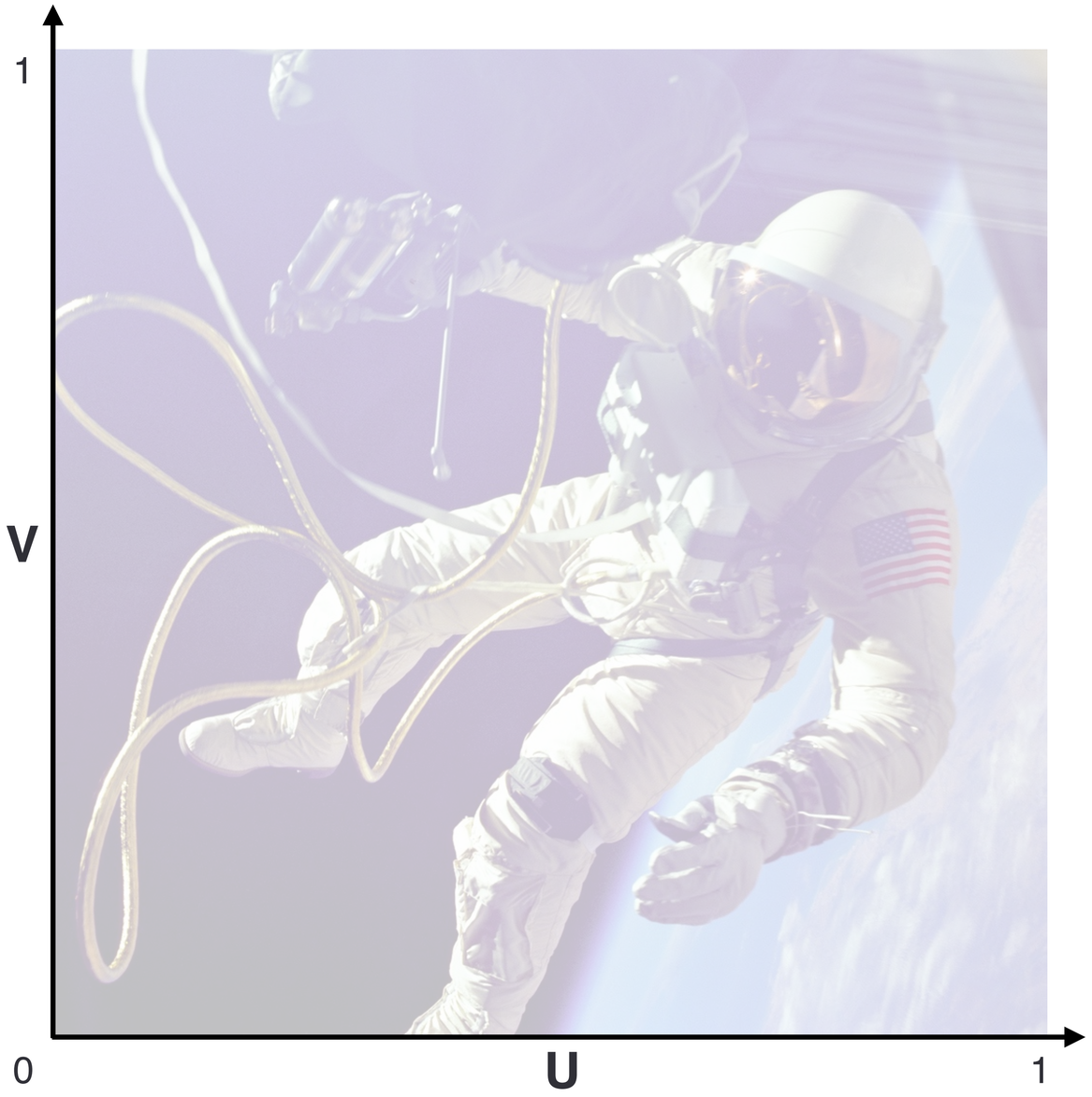
Vertices: Position



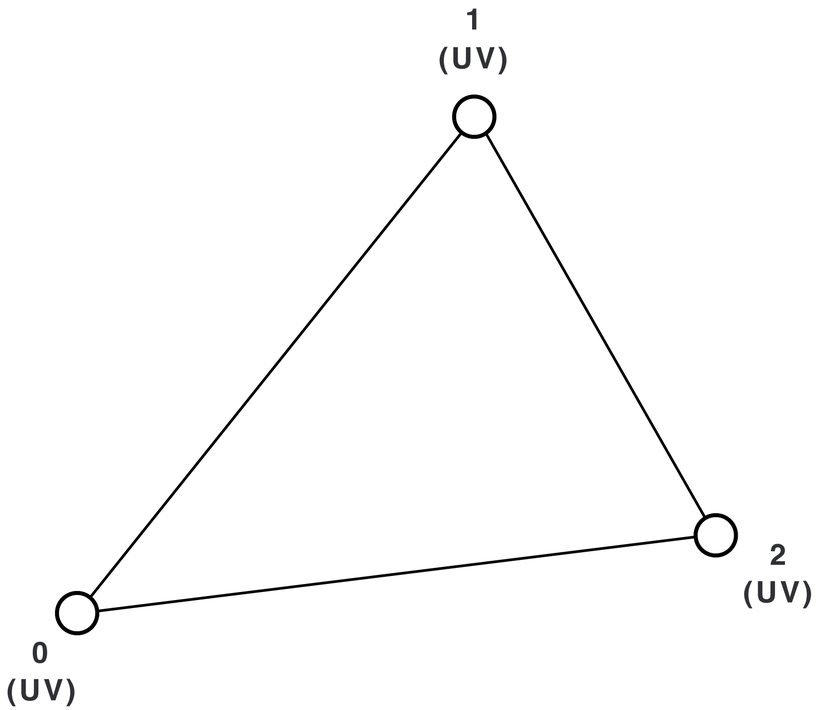
Vertices: Color

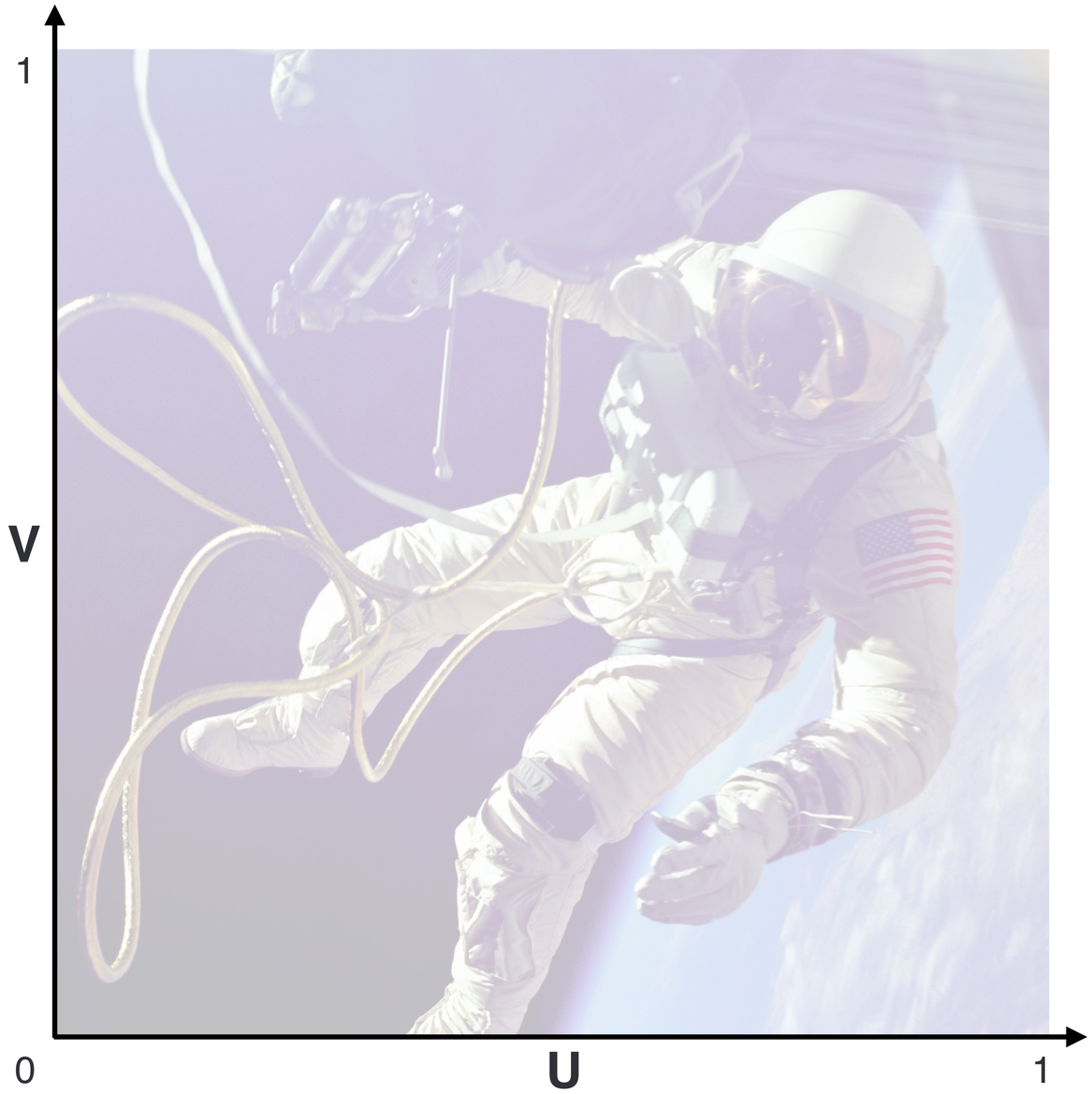


Textures: Coordinates

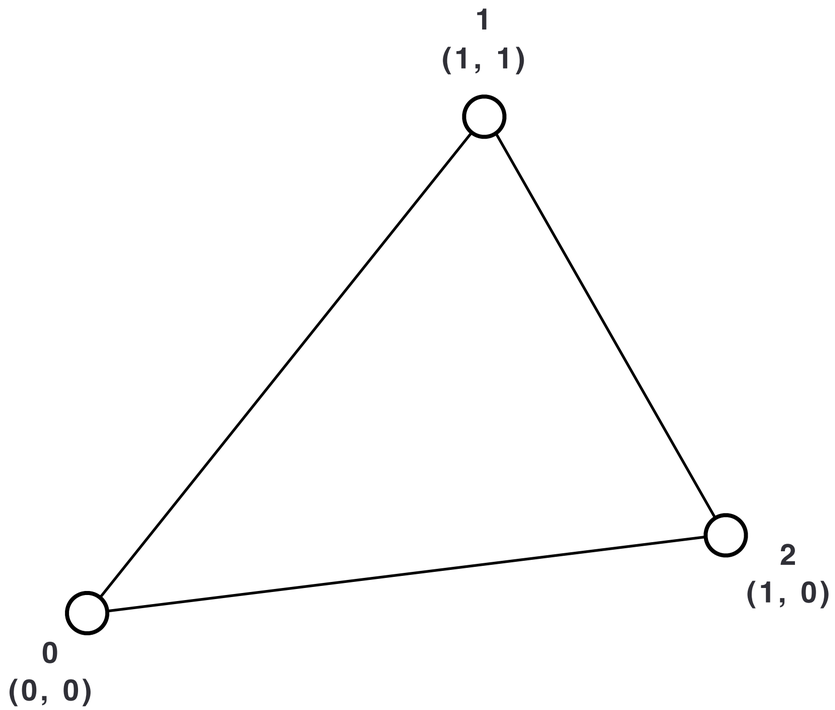


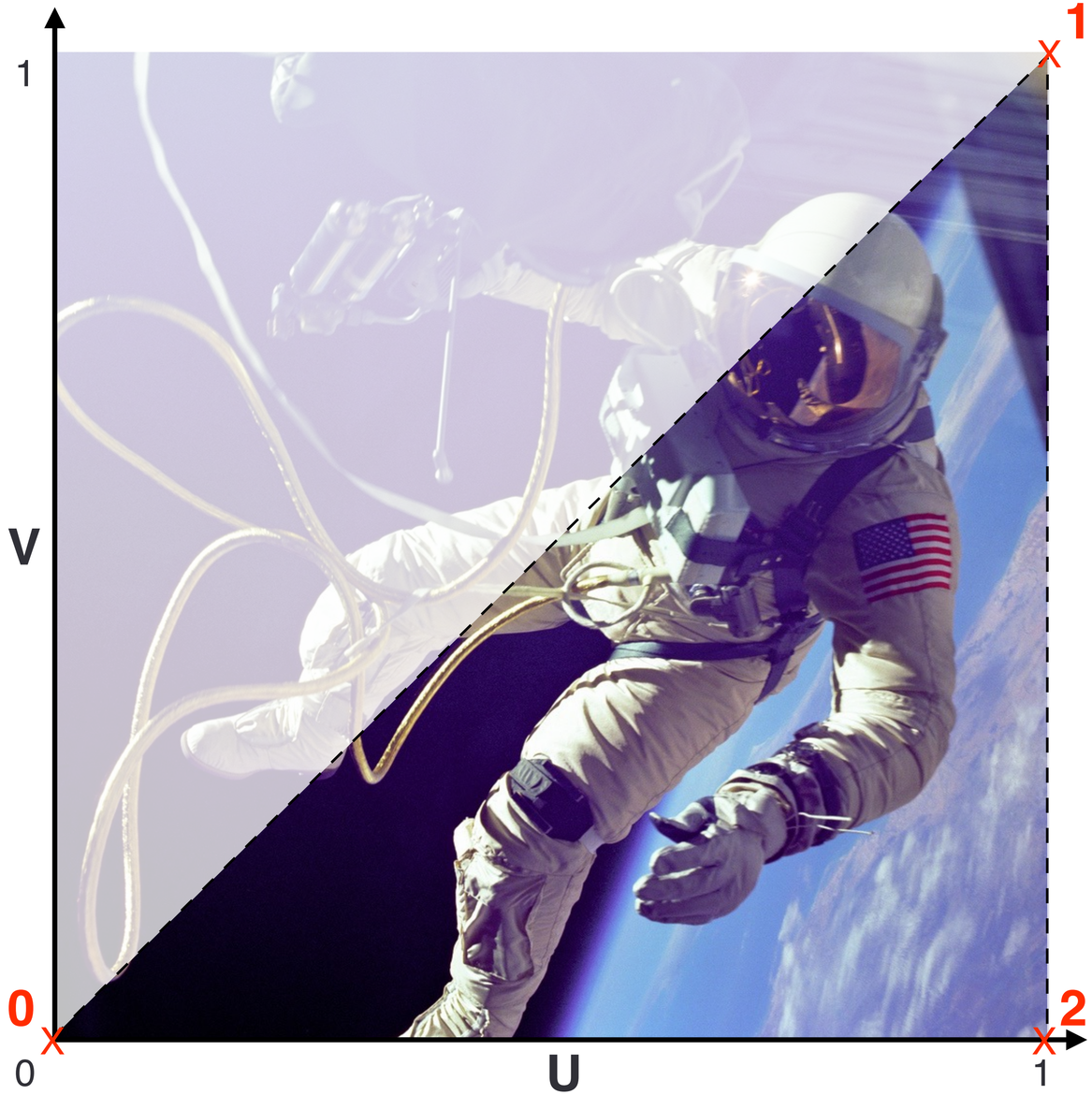
Vertices + Textures: UV Coordinates



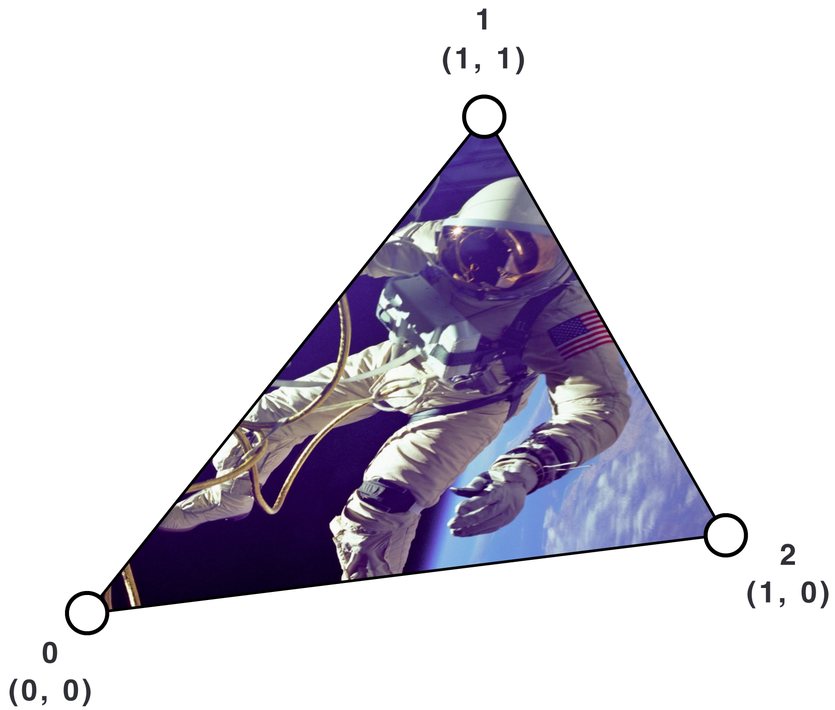


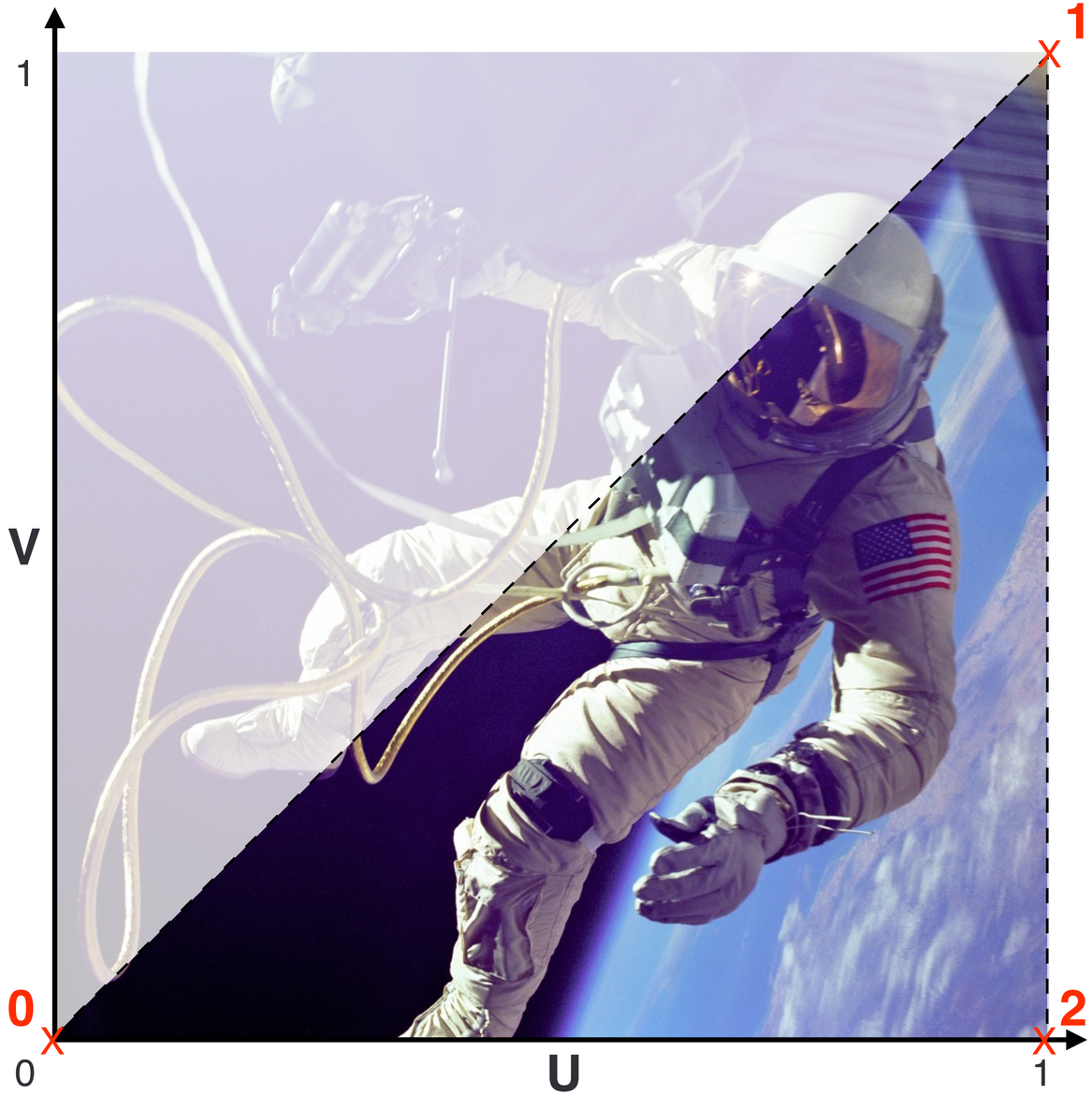
Vertices + Textures: UV Coordinates





Vertices + Textures: UV Coordinates



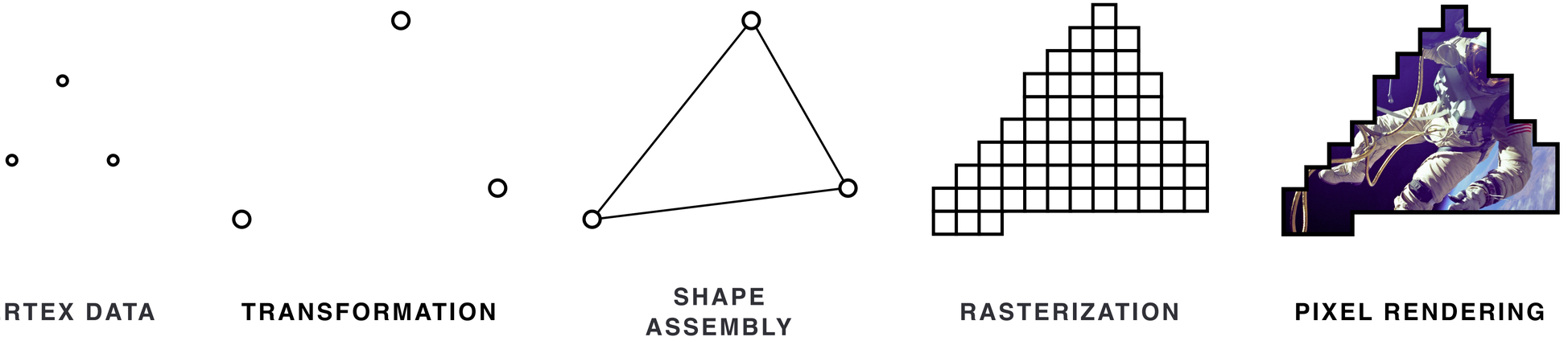


INTRO

Shader Basics

SHADER BASICS

Types of Shaders



VERTEX DATA

TRANSFORMATION

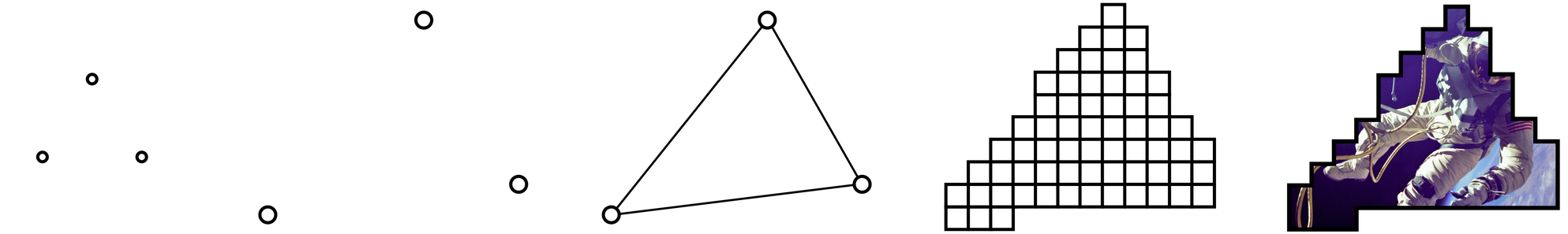
SHAPE ASSEMBLY

RASTERIZATION

PIXEL RENDERING

SHADER BASICS

Types of Shaders



VERTEX DATA

VERTEX SHADER

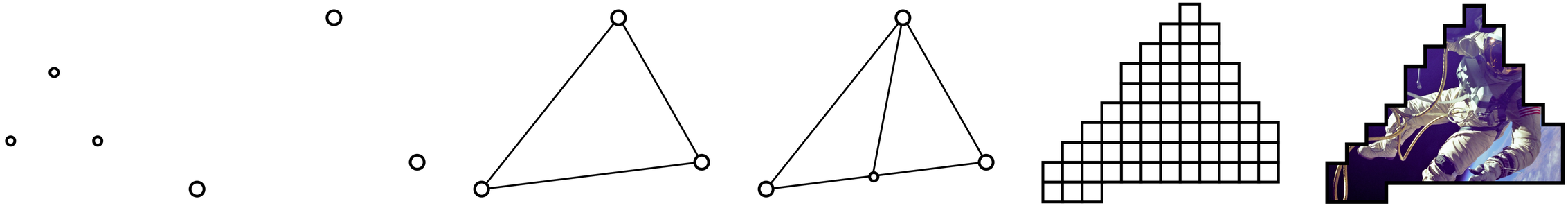
SHAPE ASSEMBLY

RASTERIZATION

FRAGMENT SHADER

SHADER BASICS

Types of Shaders



VERTEX DATA

VERTEX SHADER

SHAPE ASSEMBLY

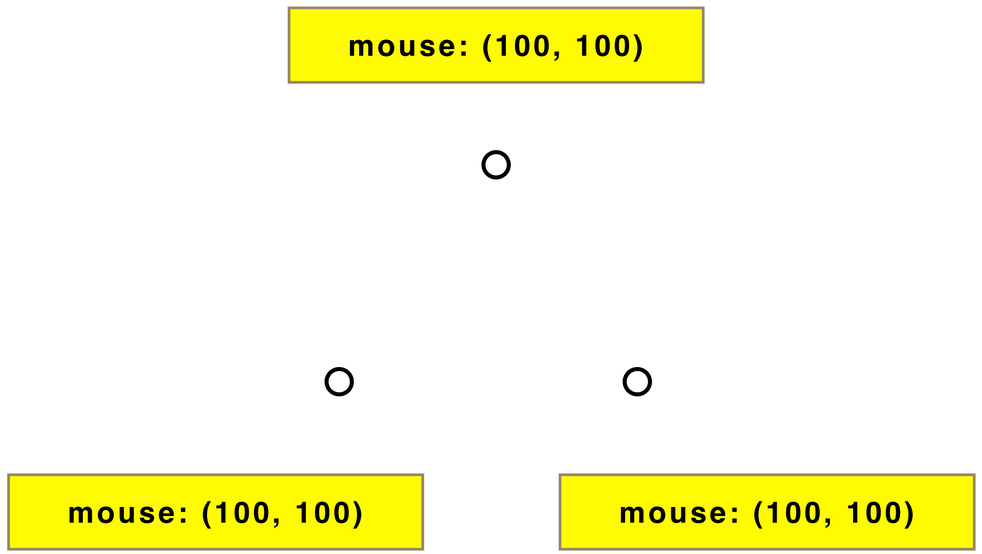
GEOMETRY SHADER

RASTERIZATION

FRAGMENT SHADER

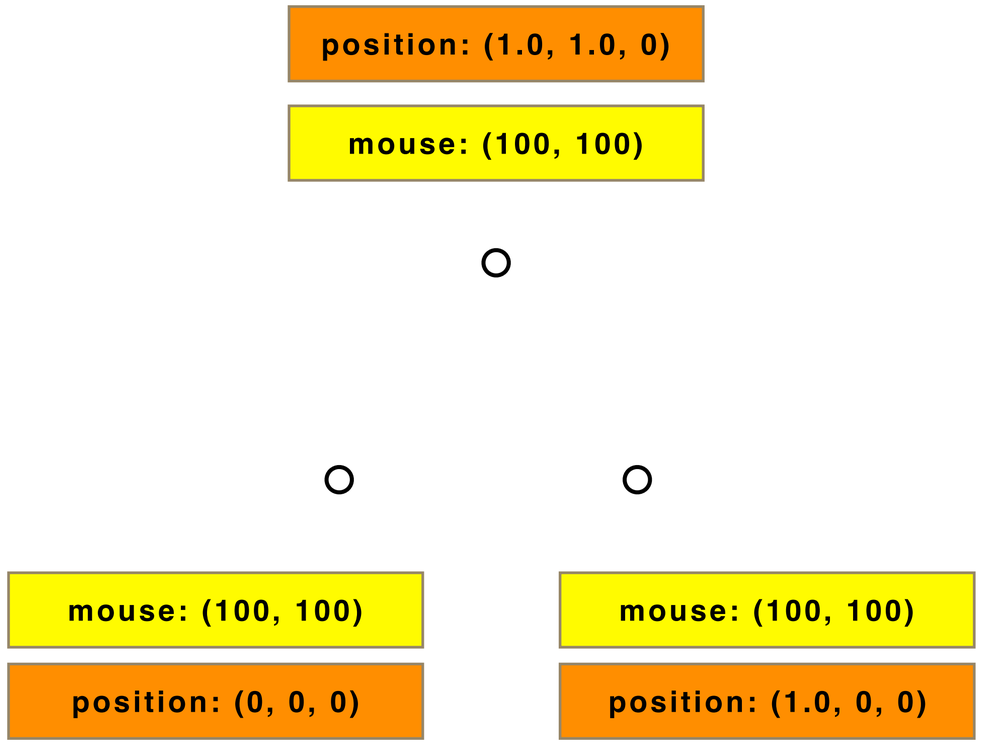
Variables: Uniforms

1. **Uniforms**
   - Read-only
   - Per primitive
     (same for all vertices & fragments)



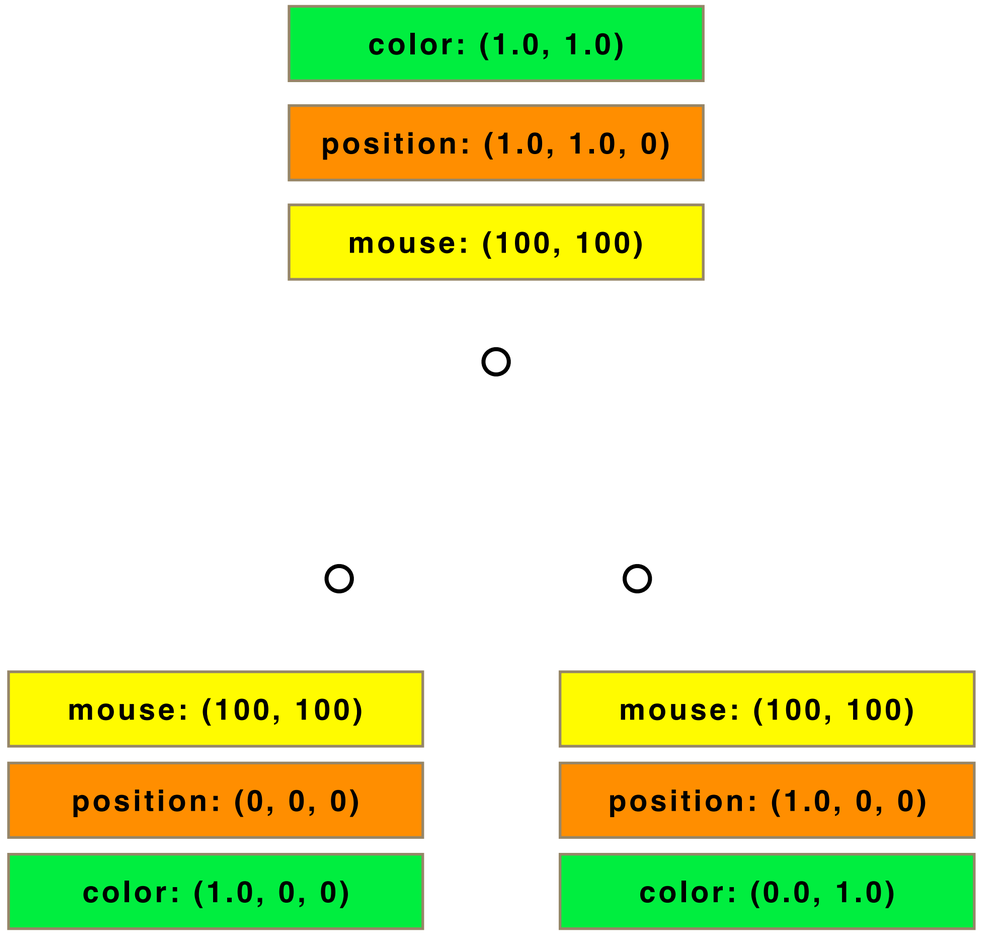
Variables: Attributes

1. **Uniforms**
   - Read-only
   - Per primitive
     (same for all vertices & fragments)
2. **Attributes**
   - Read-only
   - Per vertex

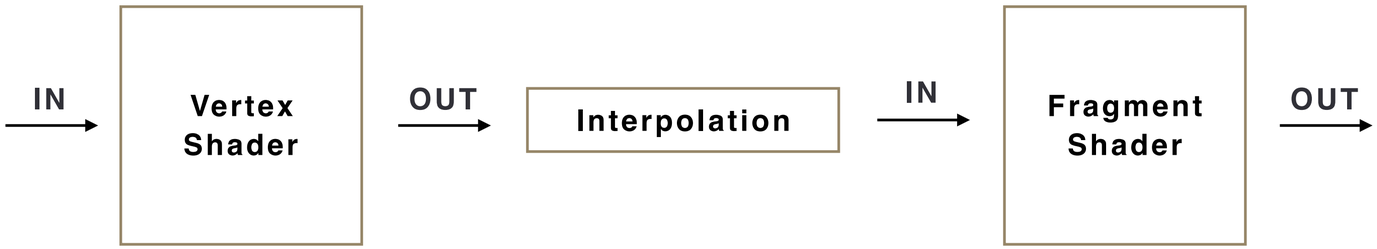


Variables: Input/Output

1. **Uniforms**
   - Read-only
   - Per primitive
     (same for all vertices & fragments)
2. **Attributes**
   - Read-only
   - Per vertex
3. **Input/Output**
   - In: Read-only
   - Out: Write-only
   - Passed from one shader stage to next

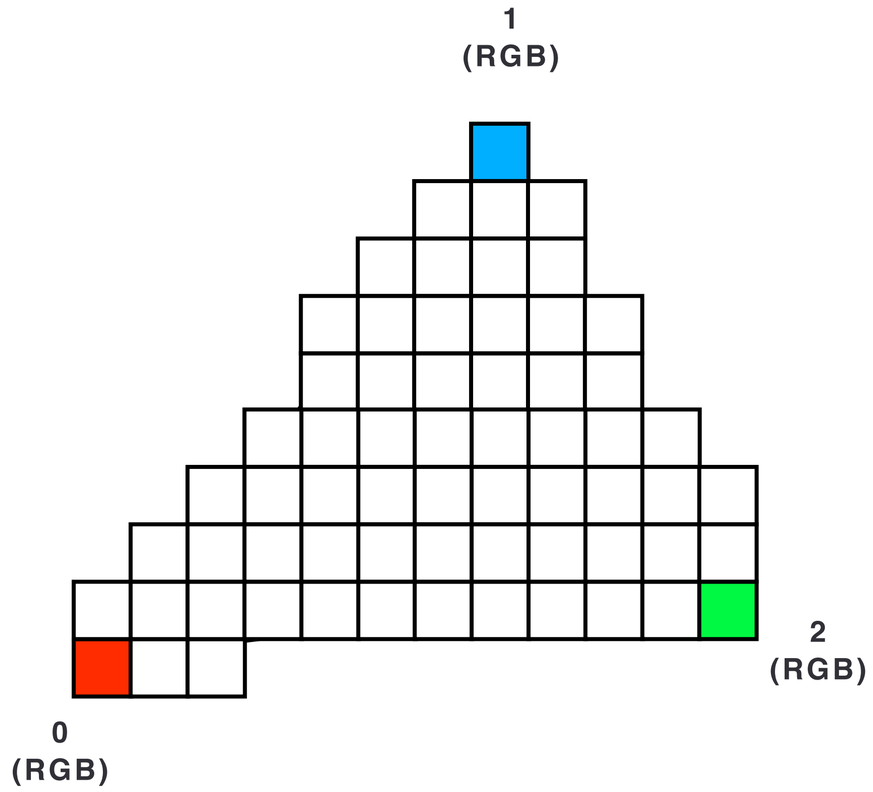


Input/Output



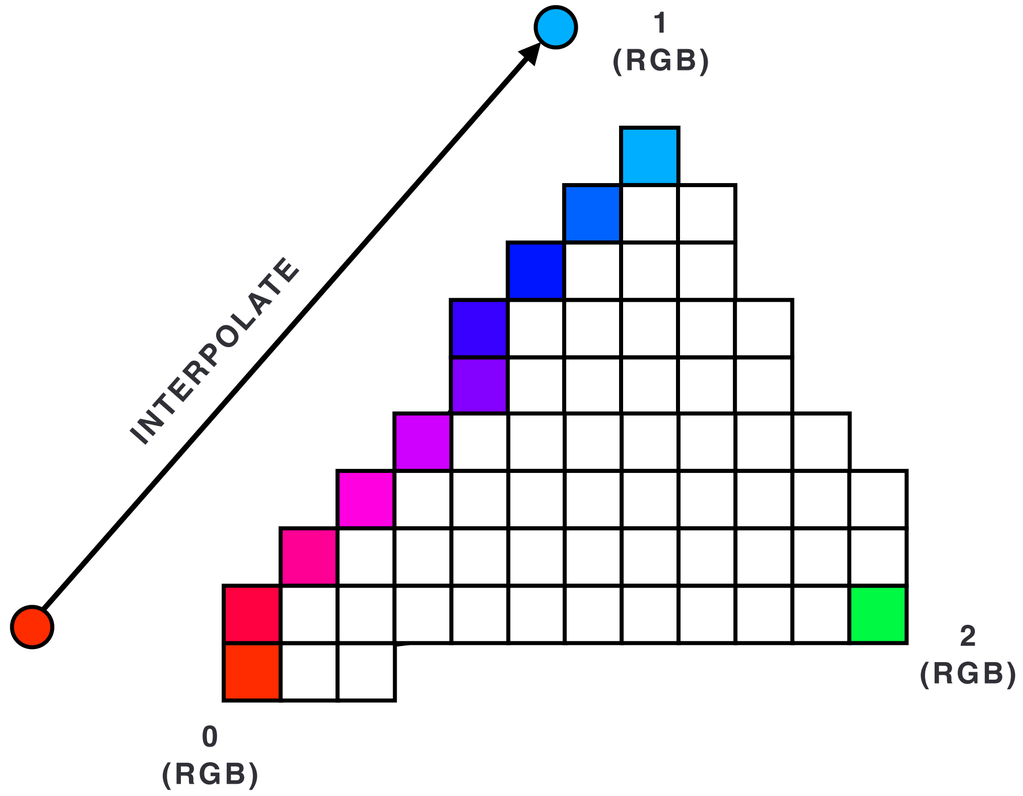
Vertices + Fragments: Interpolation

1. Per vertex color attribute as input
2. Each vertex outputs that color
3. Each fragment gets mixed color based on vertex outputs



Vertices + Fragments: Interpolation

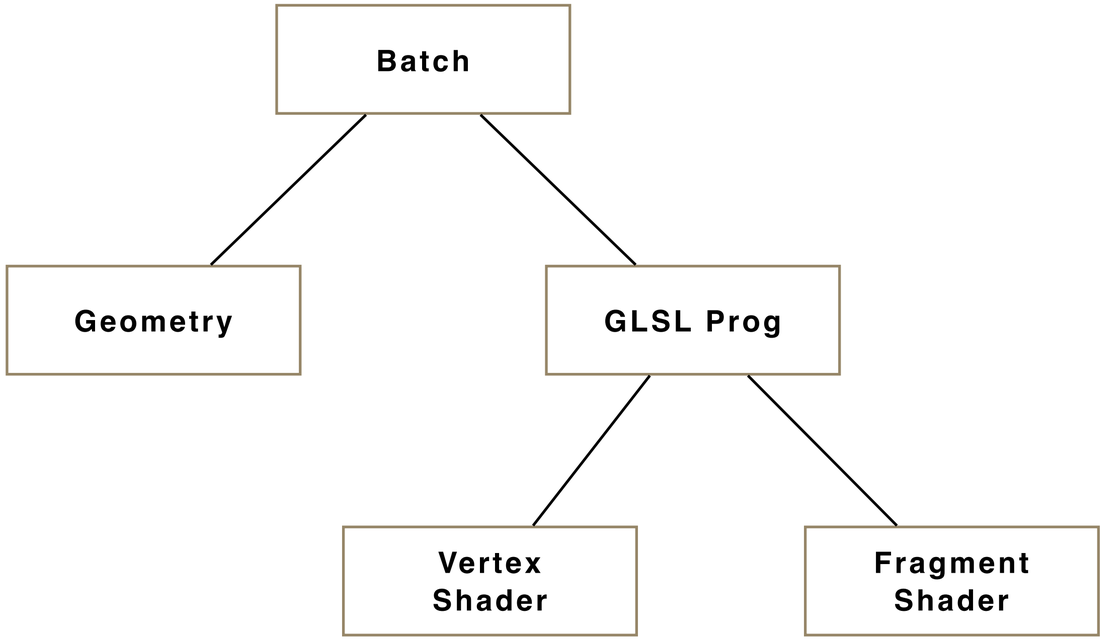
1. Per vertex color attribute as input
2. Each vertex outputs that color
3. Each fragment gets mixed color based on vertex outputs



IN PRACTICE

Shaders in Cinder

Batch Structure



Basic Example

```
void SimpleVertexShaderApp::setup() {
  auto rect = Rectf(ivec2(50), getWindowSize() - ivec2(50));
  auto geometry = geom::Rect().rect(rect);
  auto shaders = gl::GlslProg::create(
      // Vertex shader
      CI_GLSL(150,
         uniform mat4 ciModelViewProjection;
         in vec4 ciPosition;

         void main(void) {
            gl_Position = ciModelViewProjection * ciPosition;
         }
      ),
      // Fragment shader
      CI_GLSL(150,
         out vec4 oColor;

         void main(void) {
            oColor = vec4(1, 0, 0, 1);
         }
      )
  );

  mBatch = gl::Batch::create(geometry, shaders);
}

void SimpleVertexShaderApp::draw() {
  gl::clear();
  mBatch->draw();
}
```

EXAMPLE 1

Simple Vertex Shader

SIMPLE VERTEX SHADER

Moving Vertices

```
// Vertex shader
uniform float uElapsedSeconds;
uniform mat4 ciModelViewProjection;
in vec4 ciPosition;

void main(void) {
  vec4 pos = ciPosition;
  pos.x += 25.0 * cos(uElapsedSeconds);
  pos.y += 25.0 * sin(uElapsedSeconds);
  gl_Position = ciModelViewProjection * pos;
}

// ...

void SimpleVertexShaderApp::draw() {
  gl::clear();
  mBatch->getGlslProg()->uniform("uElapsedSeconds",
                                 (float)getElapsedSeconds());
  mBatch->draw();
}
```

SIMPLE VERTEX SHADER

Cinder Default Variables

UNIFORMS

- ciModelMatrix
- ciModelMatrixInverse
- ciModelMatrixInverseTranspose
- ciViewMatrix
- ciViewMatrixInverse
- ciModelView
- ciModelViewInverse
- ciModelViewInverseTranspose
- ciModelViewProjection
- ciModelViewProjectionInverse
- ciProjectionMatrix
- ciProjectionMatrixInverse
- ciViewProjection
- ciNormalMatrix
- ciViewportMatrix
- ciWindowSize
- ciElapsedSeconds

ATTRIBUTES

- ciPosition
- ciNormal
- ciTangent
- ciBitangent
- ciTexCoord0
- ciTexCoord1
- ciTexCoord2
- ciTexCoord3
- ciColor
- ciBoneIndex
- ciBoneWeight

SIMPLE VERTEX SHADER

Cinder Uniforms

```
// Vertex shader
uniform float uElapsedSeconds;
uniform mat4 ciModelViewProjection;
in vec4 ciPosition;

void main(void) {
  vec4 pos = ciPosition;
  pos.x += 25.0 * cos(uElapsedSeconds);
  pos.y += 25.0 * sin(uElapsedSeconds);
  gl_Position = ciModelViewProjection * pos;
}

// ...

void SimpleVertexShaderApp::draw() {
  gl::clear();
  mBatch->getGlslProg()->uniform("uElapsedSeconds",
                                 (float)getElapsedSeconds());
  mBatch->draw();
}
```

SIMPLE VERTEX SHADER

Cinder Uniforms

```
uniform float ciElapsedSeconds; // Use built-in Cinder uniform
uniform mat4 ciModelViewProjection;
in vec4 ciPosition;

void main(void) {
  vec4 pos = ciPosition;
  pos.x += 25.0 * cos(ciElapsedSeconds);
  pos.y += 25.0 * sin(ciElapsedSeconds);
  gl_Position = ciModelViewProjection * pos;
}

// ...

void SimpleVertexShaderApp::draw() {
  gl::clear();
  // mBatch->getGlslProg()->uniform("uElapsedSeconds",
                                    (float)getElapsedSeconds());
  mBatch->draw();
}
```

EXAMPLE 2

Mixing Colors

Vector Out, Fragment In

```
void MixingColorsApp::setup() {
  auto rect = Rectf(ivec2(0), getWindowSize());
  auto geometry = geom::Rect().rect(rect);
  auto shaders = gl::GlslProg::create(
      // Vertex shader
      CI_GLSL(150,
         uniform mat4 ciModelViewProjection;
         in vec4 ciPosition;
         out vec4 vPosition;

         void main(void) {
            vPosition = ciPosition;
            gl_Position = ciModelViewProjection * ciPosition;
         }
      ),
      // Fragment shader
      CI_GLSL(150,
         uniform vec2 uSize;
         in vec4 vPosition;
         out vec4 oColor;

         void main(void) {
            vec4 color = vec4(vPosition.x / uSize.x, 0, 0, 1);
            oColor = color;
         }
      )
  );

  mBatch = gl::Batch::create(geometry, shaders);
  mBatch->getGlslProg()->uniform("uSize", rect.getSize());
}
```

MIXING COLORS

Vector Out, Fragment In

```
void MixingColorsApp::setup() {
  auto rect = Rectf(ivec2(0), getWindowSize());
  auto geometry = geom::Rect().rect(rect);
  auto shaders = gl::GlslProg::create(
      // Vertex shader
      CI_GLSL(150,
         uniform mat4 ciModelViewProjection;
         in vec4 ciPosition;
         out vec4 vPosition;

         void main(void) {
            vPosition = ciPosition;
            gl_Position = ciModelViewProjection * ciPosition;
         }
      ),
      // Fragment shader
      CI_GLSL(150,
         uniform vec2 uSize;
         in vec4 vPosition;
         out vec4 oColor;

         void main(void) {
            vec4 color = vec4(vPosition.x / uSize.x, 0, 0, 1);
            oColor = color;
         }
      )
  );

  mBatch = gl::Batch::create(geometry, shaders);
  mBatch->getGlslProg()->uniform("uSize", rect.getSize());
}
```

MIXING COLORS

Vector Out, Fragment In

```
void MixingColorsApp::setup() {
  auto rect = Rectf(ivec2(0), getWindowSize());
  auto geometry = geom::Rect().rect(rect);
  auto shaders = gl::GlslProg::create(
      // Vertex shader
      CI_GLSL(150,
         uniform mat4 ciModelViewProjection;
         in vec4 ciPosition;
         out vec4 vPosition;

         void main(void) {
            vPosition = ciPosition;
            gl_Position = ciModelViewProjection * ciPosition;
         }
      ),
      // Fragment shader
      CI_GLSL(150,
         uniform vec2 uSize;
         in vec4 vPosition;
         out vec4 oColor;

         void main(void) {
            vec4 color = vec4(vPosition.x / uSize.x, 0, 0, 1);
            oColor = color;
         }
      )
  );

  mBatch = gl::Batch::create(geometry, shaders);
  mBatch->getGlslProg()->uniform("uSize", rect.getSize());
}
```

MIXING COLORS

Vector Out, Fragment In

```
void MixingColorsApp::setup() {
  auto rect = Rectf(ivec2(0), getWindowSize());
  auto geometry = geom::Rect().rect(rect);
  auto shaders = gl::GlslProg::create(
      // Vertex shader
      CI_GLSL(150,
         uniform mat4 ciModelViewProjection;
         in vec4 ciPosition;
         out vec4 vPosition;

         void main(void) {
            vPosition = ciPosition;
            gl_Position = ciModelViewProjection * ciPosition;
         }
      ),
      // Fragment shader
      CI_GLSL(150,
         uniform vec2 uSize;
         in vec4 vPosition;
         out vec4 oColor;

         void main(void) {
            vec4 color = vec4(vPosition.x / uSize.x, 0, 0, 1);
            oColor = color;
         }
      )
  );

  mBatch = gl::Batch::create(geometry, shaders);
  mBatch->getGlslProg()->uniform("uSize", rect.getSize());
}
```

MIXING COLORS

Vector Out, Fragment In

```
void MixingColorsApp::setup() {
  auto rect = Rectf(ivec2(0), getWindowSize());
  auto geometry = geom::Rect().rect(rect);
  auto shaders = gl::GlslProg::create(
      // Vertex shader
      CI_GLSL(150,
         uniform mat4 ciModelViewProjection;
         in vec4 ciPosition;
         out vec4 vPosition;

         void main(void) {
            vPosition = ciPosition;
            gl_Position = ciModelViewProjection * ciPosition;
         }
      ),
      // Fragment shader
      CI_GLSL(150,
         uniform vec2 uSize;
         in vec4 vPosition;
         out vec4 oColor;

         void main(void) {
            vec4 color = vec4(vPosition.x / uSize.x, 0, 0, 1);
            oColor = color;
         }
      )
  );

  mBatch = gl::Batch::create(geometry, shaders);
  mBatch->getGlslProg()->uniform("uSize", rect.getSize());
}
```

MIXING COLORS

Vector Out, Fragment In

```
void MixingColorsApp::setup() {
  auto rect = Rectf(ivec2(0), getWindowSize());
  auto geometry = geom::Rect().rect(rect);
  auto shaders = gl::GlslProg::create(
      // Vertex shader
      CI_GLSL(150,
         uniform mat4 ciModelViewProjection;
         in vec4 ciPosition;
         out vec4 vPosition;

         void main(void) {
            vPosition = ciPosition;
            gl_Position = ciModelViewProjection * ciPosition;
         }
      ),
      // Fragment shader
      CI_GLSL(150,
         uniform vec2 uSize;
         in vec4 vPosition;
         out vec4 oColor;

         void main(void) {
            vec4 color = vec4(vPosition.x / uSize.x,
                              vPosition.y / uSize.y, 0, 1);
            oColor = color;
         }
      )
  );

  mBatch = gl::Batch::create(geometry, shaders);
  mBatch->getGlslProg()->uniform("uSize", rect.getSize());
}
```

MIXING COLORS

Vertex Attribute Colors

```
auto rect = Rectf(ivec2(0), getWindowSize());
auto geometry = geom::Rect()
   .rect(rect)
   .colors(ColorAf(0, 0, 0, 1), ColorAf(1, 0, 0, 1),
           ColorAf(1, 1, 0, 1), ColorAf(0, 1, 0, 1));

auto shaders = gl::GlslProg::create(
  // Vertex shader
  CI_GLSL(150,
   uniform mat4 ciModelViewProjection;
   in vec4 ciPosition;
   in vec4 ciColor;
   out vec4 vColor;

   void main(void) {
      vColor = ciColor;
      gl_Position = ciModelViewProjection * ciPosition;
   }
  ),
  // Fragment shader
  CI_GLSL(150,
   in vec4 vColor;
   out vec4 oColor;

   void main(void) {
      oColor = vColor;
   }
  )
);

mBatch = gl::Batch::create(geometry, shaders);
```

MIXING COLORS

Vertex Attribute Colors

```
auto rect = Rectf(ivec2(0), getWindowSize());
auto geometry = geom::Rect()
   .rect(rect)
   .colors(ColorAf(0, 0, 0, 1), ColorAf(1, 0, 0, 1),
           ColorAf(1, 1, 0, 1), ColorAf(0, 1, 0, 1));

auto shaders = gl::GlslProg::create(
  // Vertex shader
  CI_GLSL(150,
   uniform mat4 ciModelViewProjection;
   in vec4 ciPosition;
   in vec4 ciColor;
   out vec4 vColor;

   void main(void) {
      vColor = ciColor;
      gl_Position = ciModelViewProjection * ciPosition;
   }
  ),
  // Fragment shader
  CI_GLSL(150,
   in vec4 vColor;
   out vec4 oColor;

   void main(void) {
      oColor = vColor;
   }
  )
);

mBatch = gl::Batch::create(geometry, shaders);
```

MIXING COLORS

Vertex Attribute Colors

```
auto rect = Rectf(ivec2(0), getWindowSize());
auto geometry = geom::Rect()
   .rect(rect)
   .colors(ColorAf(0, 0, 0, 1), ColorAf(1, 0, 0, 1),
           ColorAf(1, 1, 0, 1), ColorAf(0, 1, 0, 1));

auto shaders = gl::GlslProg::create(
  // Vertex shader
  CI_GLSL(150,
   uniform mat4 ciModelViewProjection;
   in vec4 ciPosition;
   in vec4 ciColor;
   out vec4 vColor;

   void main(void) {
      vColor = ciColor;
      gl_Position = ciModelViewProjection * ciPosition;
   }
  ),
  // Fragment shader
  CI_GLSL(150,
   in vec4 vColor;
   out vec4 oColor;

   void main(void) {
      oColor = vColor;
   }
  )
);

mBatch = gl::Batch::create(geometry, shaders);
```

MIXING COLORS

Vertex Attribute Interpolation

```
auto rect = Rectf(ivec2(0), getWindowSize());
auto geometry = geom::Rect()
   .rect(rect)
   .colors(ColorAf(0, 0, 0, 1), ColorAf(1, 0, 0, 1),
           ColorAf(1, 1, 0, 1), ColorAf(0, 1, 0, 1));

auto shaders = gl::GlslProg::create(
  // Vertex shader
  CI_GLSL(150,
   uniform mat4 ciModelViewProjection;
   in vec4 ciPosition;
   in vec4 ciColor;
   out vec4 vColor;

   void main(void) {
      vColor = ciColor;
      gl_Position = ciModelViewProjection * ciPosition;
   }
  ),
  // Fragment shader
  CI_GLSL(150,
   in vec4 vColor;
   out vec4 oColor;

   void main(void) {
      oColor = vColor;
   }
  )
);

mBatch = gl::Batch::create(geometry, shaders);
```

MIXING COLORS

Cinder Stock Shaders

```
auto rect = Rectf(ivec2(0), getWindowSize());
auto geometry = geom::Rect()
   .rect(rect)
   .colors(ColorAf(0, 0, 0, 1), ColorAf(1, 0, 0, 1),
           ColorAf(1, 1, 0, 1), ColorAf(0, 1, 0, 1));
auto shaders = gl::getStockShader(gl::ShaderDef().color());
mBatch = gl::Batch::create(geometry, shaders);
```

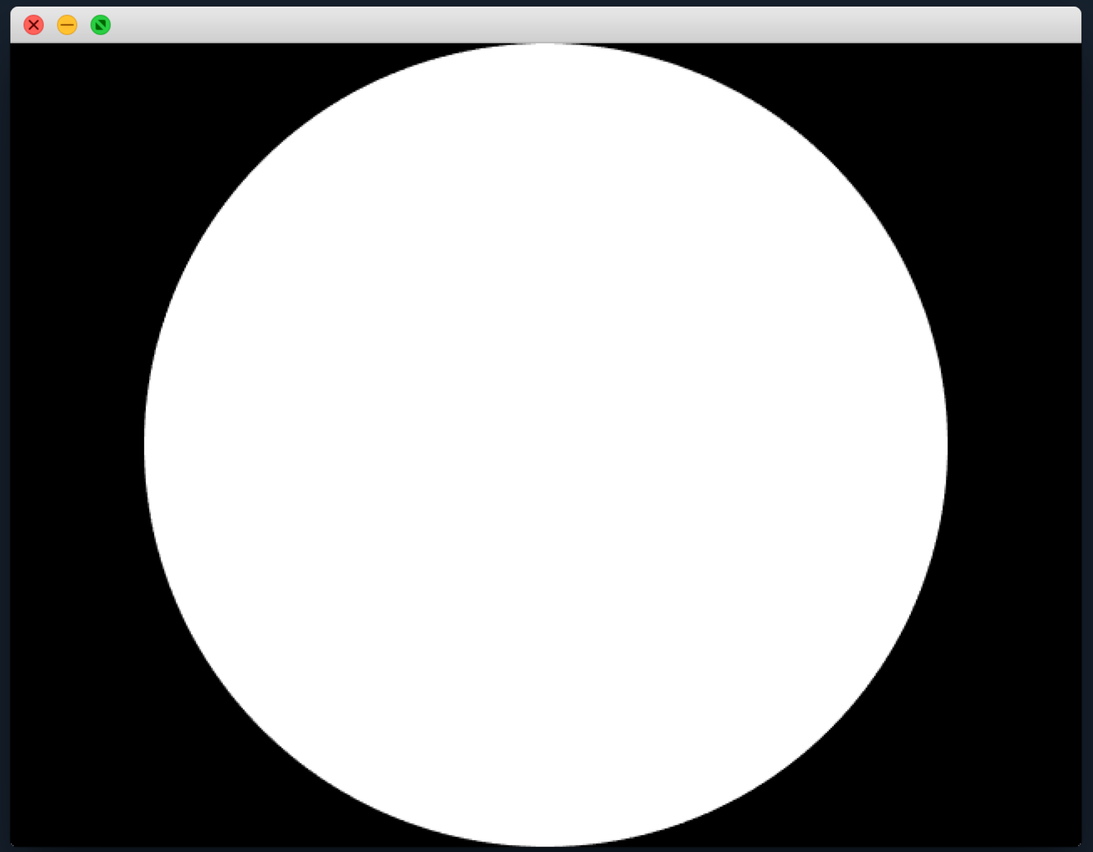
EXAMPLE 3

Drawing A Circle

DRAWING A CIRCLE

Circle Geometry

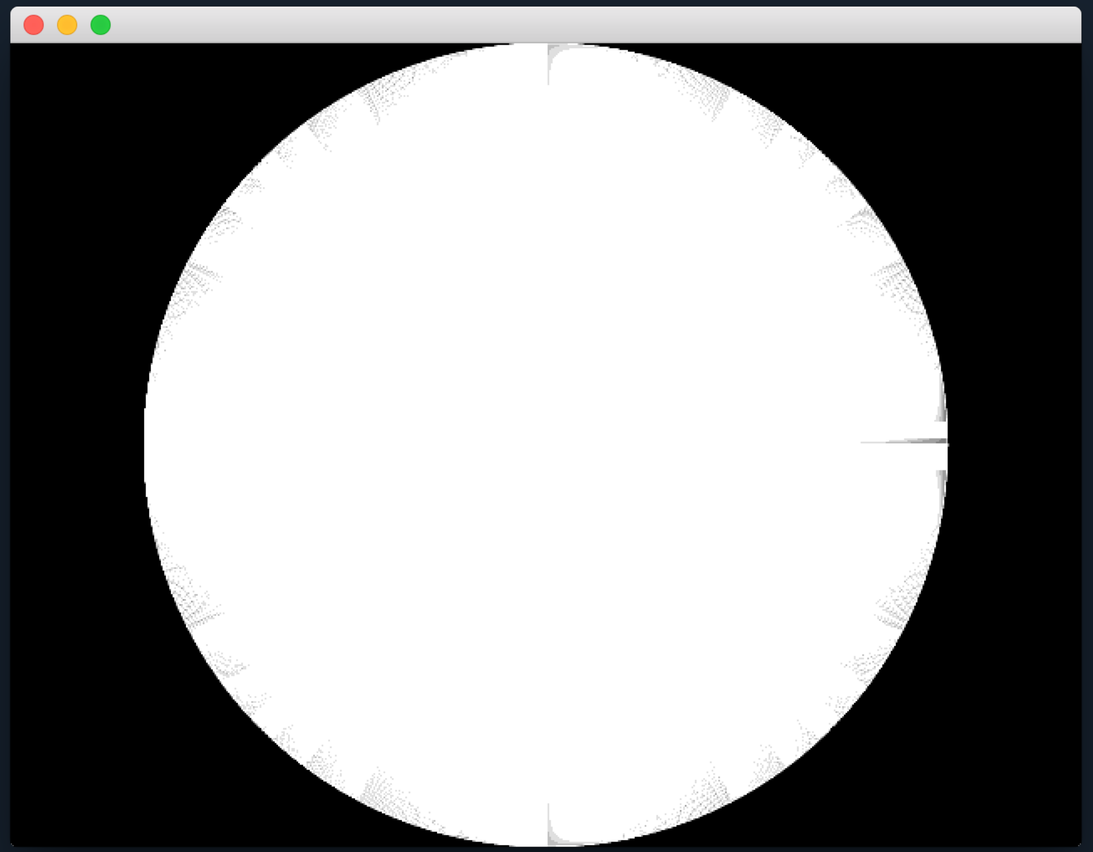
```
auto geometry = geom::Circle()
  .radius(getWindowHeight() * 0.5f)
  .center(vec2(getWindowSize()) * 0.5f);

auto shaders = gl::getStockShader(gl::ShaderDef().color());

mBatch = gl::Batch::create(geometry, shaders);
```



DRAWING A CIRCLE

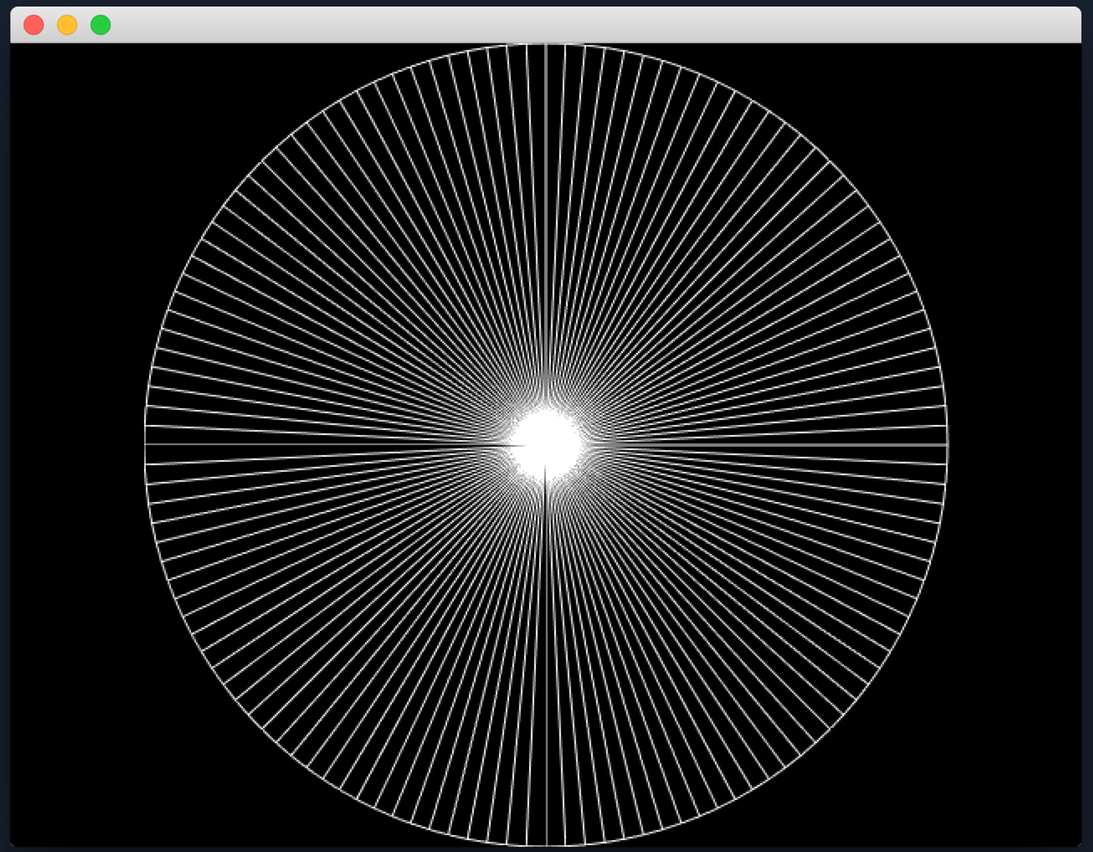
Circle Geometry

```
auto geometry = geom::Circle()
  .radius(getWindowHeight() * 0.5f)
  .center(vec2(getWindowSize()) * 0.5f);

auto shaders = gl::getStockShader(gl::ShaderDef().color());

mBatch = gl::Batch::create(geometry, shaders);

gl::enableWireframe();
```



Circle Geometry

```
auto geometry = geom::Circle()
  .subdivisions(128)
  .radius(getWindowHeight() * 0.5f)
  .center(vec2(getWindowSize()) * 0.5f);

auto shaders = gl::getStockShader(gl::ShaderDef().color());

mBatch = gl::Batch::create(geometry, shaders);

gl::enableWireframe();
```



DRAWING A CIRCLE

There has to be a better way…

DRAWING A CIRCLE

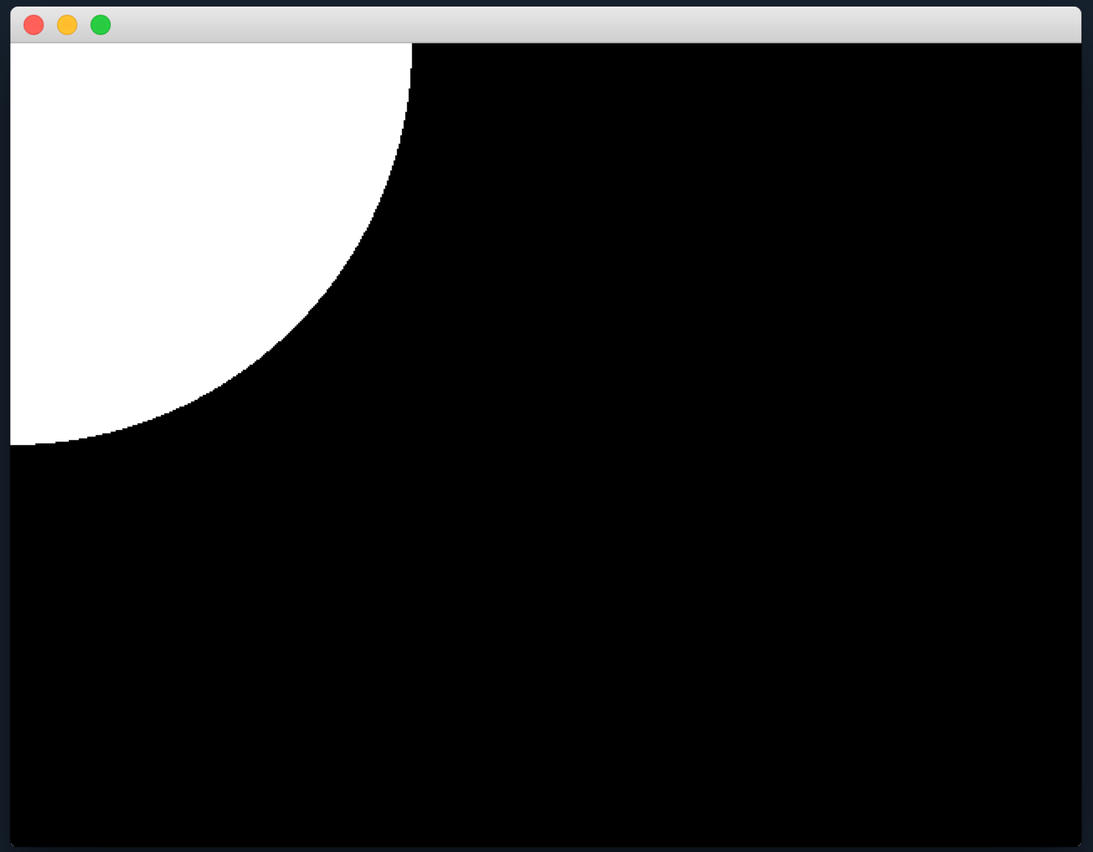
Circle Geometry

```
auto rect = Rectf(ivec2(0), getWindowSize());
auto geometry = geom::Rect().rect(rect);
auto shaders = gl::GlslProg::create(
  // Vertex shader
  CI_GLSL(150,
   uniform mat4 ciModelViewProjection;
   in vec4 ciPosition;
   void main(void) {
      gl_Position = ciModelViewProjection * ciPosition;
   }
  ),
  // Fragment shader
  CI_GLSL(150,
   out vec4 oColor;

   void main(void) {
      oColor = vec4(1, 1, 1, 1);
   }
  )
);

mBatch = gl::Batch::create(geometry, shaders);
```

DRAWING A CIRCLE

Circle Geometry

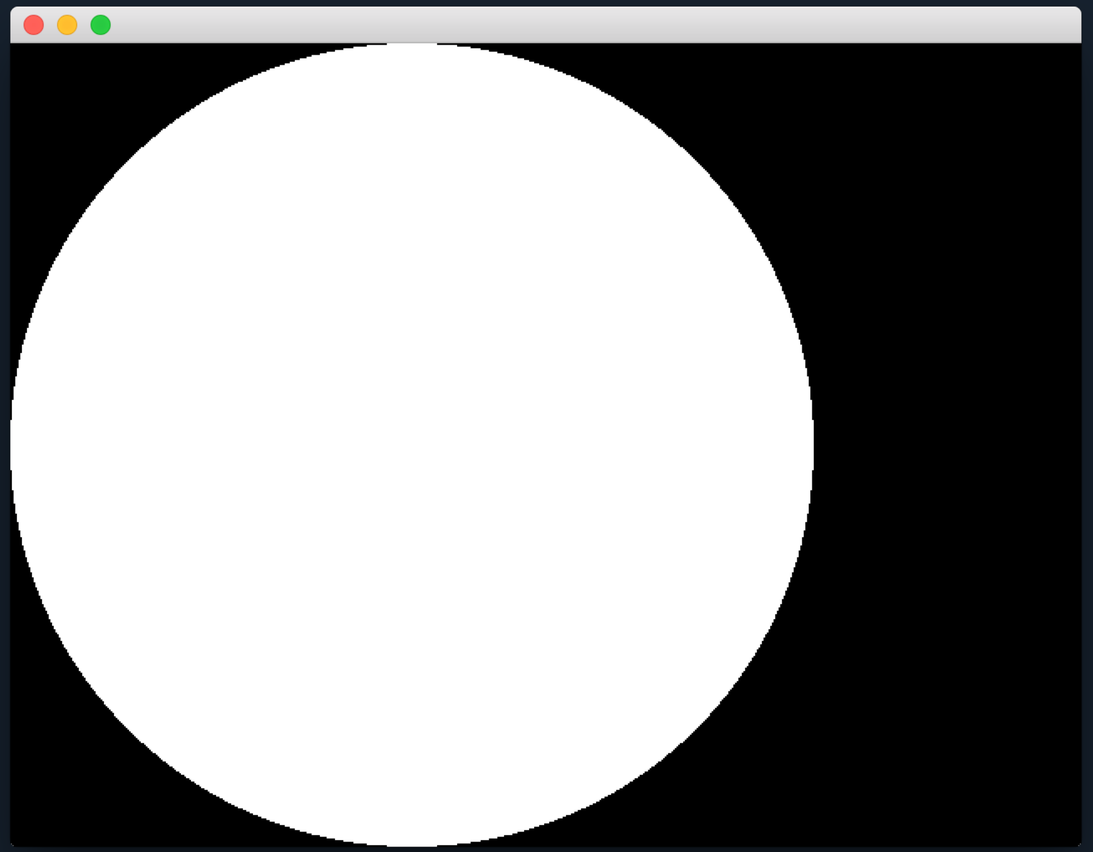
```
auto shaders = gl::GlslProg::create(
  // Vertex shader
  CI_GLSL(150,
   uniform mat4 ciModelViewProjection;
   in vec4 ciPosition;
   out vec4 vPosition;
   void main(void) {
      vPosition = ciPosition;
      gl_Position = ciModelViewProjection * ciPosition;
   }
  ),
  // Fragment shader
  CI_GLSL(150,
   uniform float uRadius;
   out vec4 oColor;
   in vec4 vPosition;

   void main(void) {
      if (length(vPosition.xy) > uRadius) {
         discard;
      }
      oColor = vec4(1, 1, 1, 1);
   }
  )
);

mBatch = gl::Batch::create(geometry, shaders);
mBatch->getGlslProg()->uniform("uRadius", (float)getWindowHeight() * 0.5f);
```



DRAWING A CIRCLE

Circle Geometry

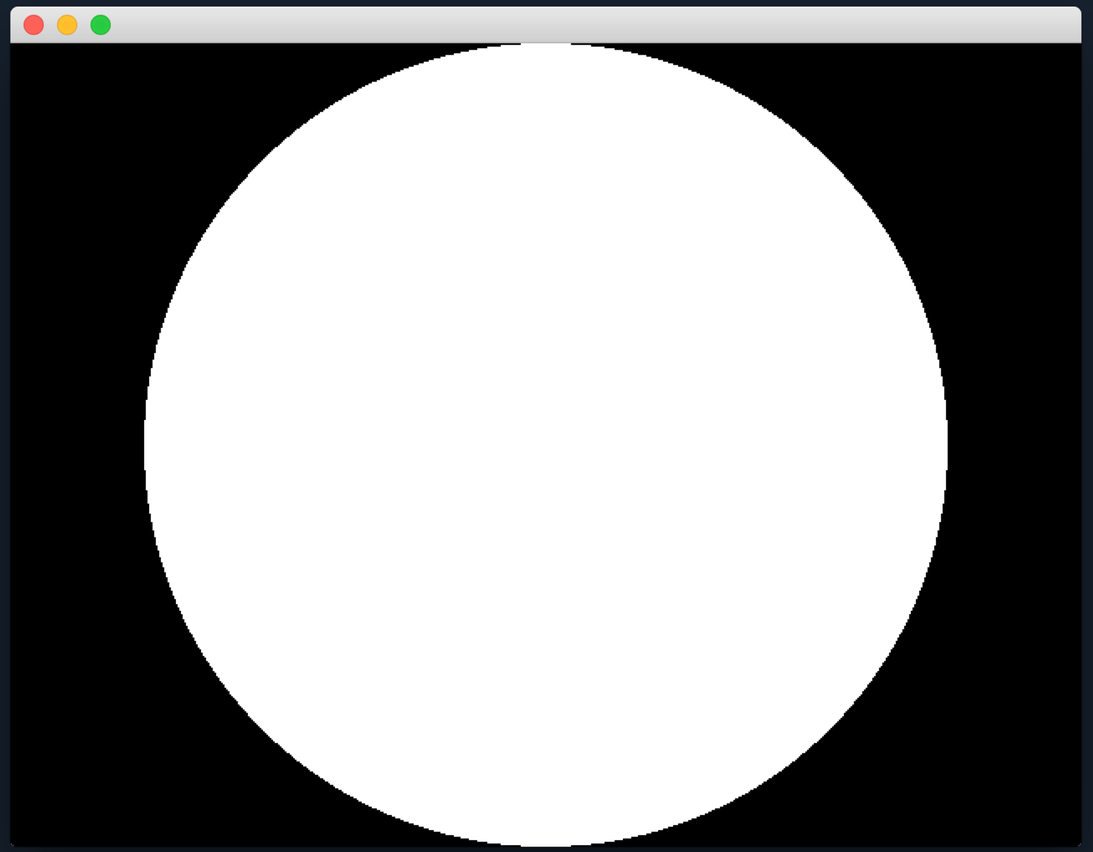
```
auto shaders = gl::GlslProg::create(
  // Vertex shader
  CI_GLSL(150,
   uniform mat4 ciModelViewProjection;
   in vec4 ciPosition;
   out vec4 vPosition;
   void main(void) {
      vPosition = ciPosition;
      gl_Position = ciModelViewProjection * ciPosition;
   }
  ),
  // Fragment shader
  CI_GLSL(150,
   uniform float uRadius;
   out vec4 oColor;
   in vec4 vPosition;

   void main(void) {
      if (length(vPosition.xy - vec2(uRadius)) > uRadius) {
         discard;
      }
      oColor = vec4(1, 1, 1, 1);
   }
  )
);

mBatch = gl::Batch::create(geometry, shaders);
mBatch->getGlslProg()->uniform("uRadius", (float)getWindowHeight() * 0.5f);
```



DRAWING A CIRCLE

Circle Geometry

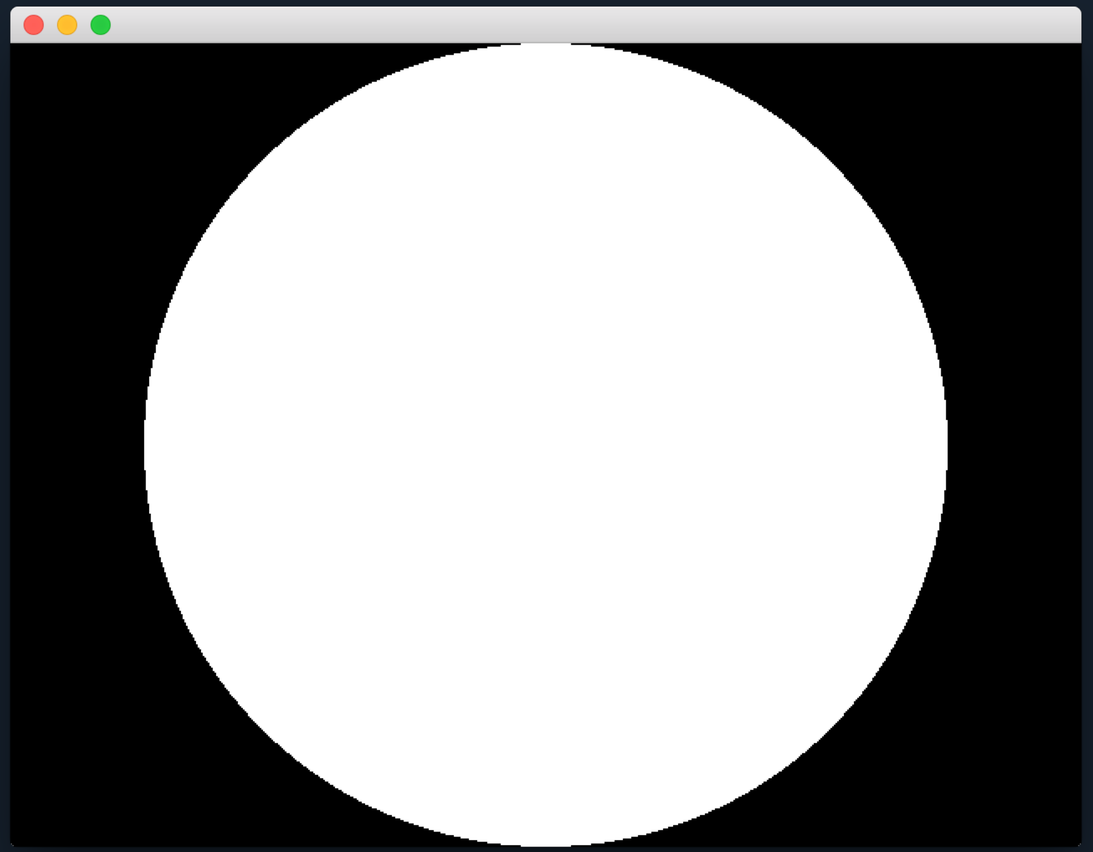
```
auto shaders = gl::GlslProg::create(
  // Vertex shader
  CI_GLSL(150,
   uniform mat4 ciModelViewProjection;
   in vec4 ciPosition;
   out vec4 vPosition;
   void main(void) {
      vPosition = ciPosition;
      gl_Position = ciModelViewProjection * ciPosition;
   }
  ),
  // Fragment shader
  CI_GLSL(150,
   uniform float uRadius;
   uniform vec2 uSize;
   out vec4 oColor;
   in vec4 vPosition;

   void main(void) {
      if (length(vPosition.xy - uSize * 0.5) > uRadius) {
         discard;
      }
      oColor = vec4(1, 1, 1, 1);
   }
  )
);

mBatch = gl::Batch::create(geometry, shaders);
mBatch->getGlslProg()->uniform("uSize", vec2(getWindowSize()));
mBatch->getGlslProg()->uniform("uRadius", (float)getWindowHeight() * 0.5f);
```



DRAWING A CIRCLE

Circle Geometry

```
auto rect = Rectf(ivec2(0), getWindowSize());
auto geometry = geom::Rect().rect(rect);
auto shaders = gl::GlslProg::create(
  // Vertex shader
  CI_GLSL(150,
   uniform mat4 ciModelViewProjection;
   in vec4 ciPosition;
   out vec4 vPosition;
   void main(void) {
      vPosition = ciPosition;
      gl_Position = ciModelViewProjection * ciPosition;
   }
  ),
  // Fragment shader
  CI_GLSL(150,
   uniform vec2 uSize;
   out vec4 oColor;
   in vec4 vPosition;

   void main(void) {
      float radius = min(uSize.x, uSize.y) * 0.5f;
      if (length(vPosition.xy - uSize * 0.5) > radius) {
         discard;
      }
      oColor = vec4(1, 1, 1, 1);
   }
  )
);

mBatch = gl::Batch::create(geometry, shaders);
mBatch->getGlslProg()->uniform("uSize", vec2(getWindowSize()));
```

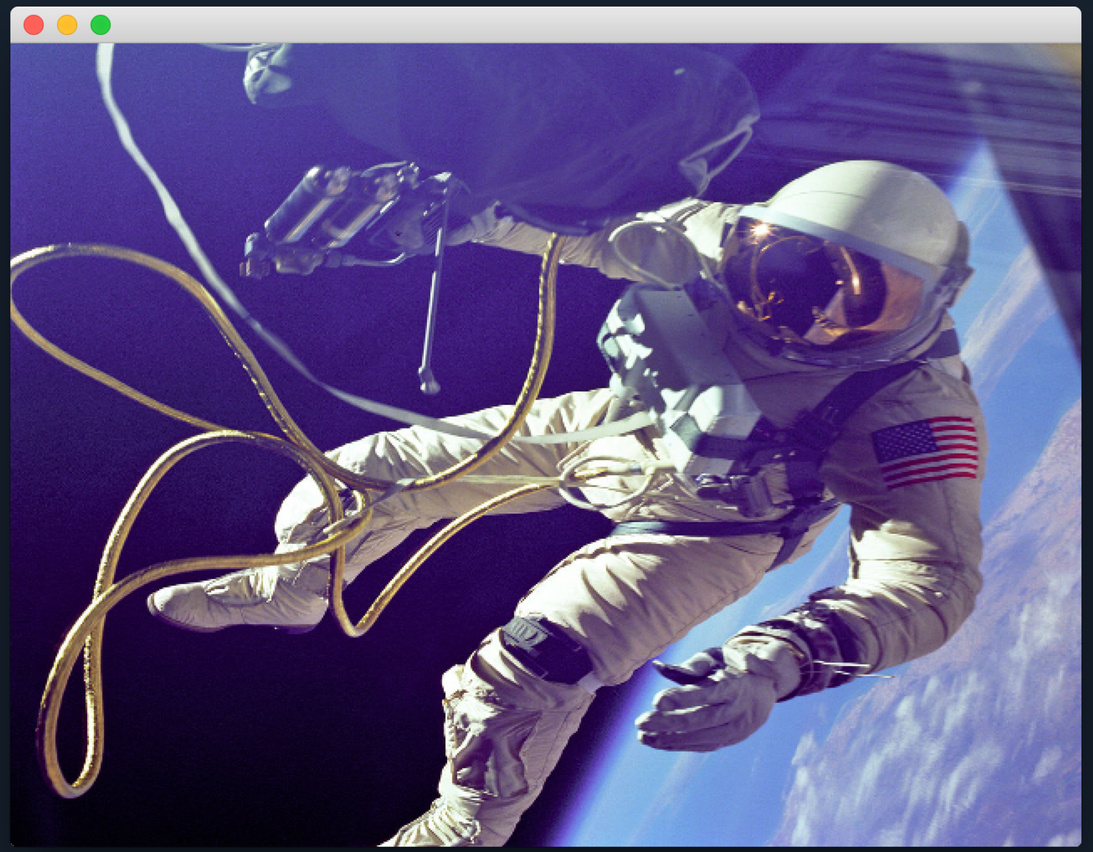


EXAMPLE 4

Blending Textures

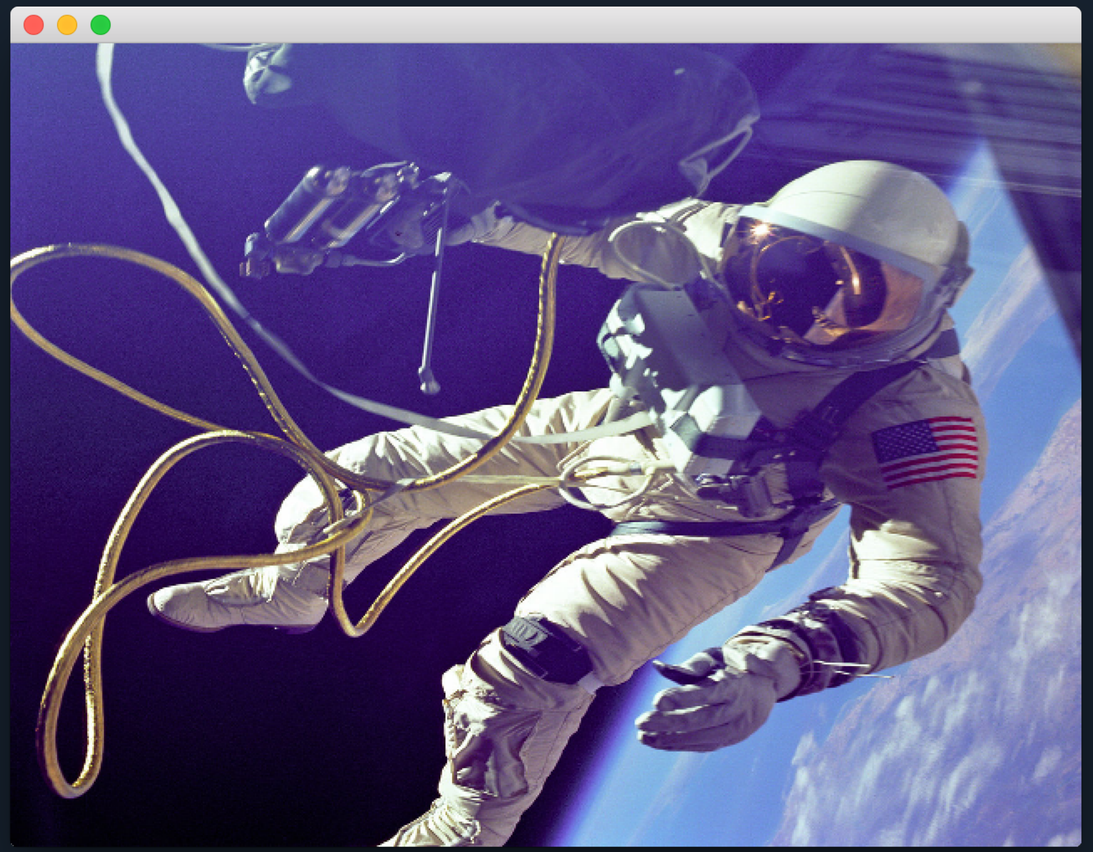
Stock Shader

```
class BlendingTexturesApp : public App {
public:
  void setup() override;
  void draw() override;

protected:
  gl::BatchRef mBatch;
  gl::Texture2dRef mTexture;
};

void BlendingTexturesApp::setup() {
  auto rect = Rectf(ivec2(0), getWindowSize());
  auto geometry = geom::Rect().rect(rect);
  auto shaders = gl::getStockShader(gl::ShaderDef().texture());
  mBatch = gl::Batch::create(geometry, shaders);

  auto image = loadImage(loadAsset("texture01.jpg"));
  mTexture = gl::Texture2d::create(image);
  mTexture->bind(); // set as the currently bound texture
}

void BlendingTexturesApp::draw() {
  gl::clear();
  mBatch->draw();
}
```



Custom Shader

```
auto shaders = gl::GlslProg::create(
  // Vertex shader
  CI_GLSL(150,
   uniform mat4 ciModelViewProjection;
   in vec4 ciPosition;
   in vec2 ciTexCoord0;
   out vec2 vTexCoord0;

   void main(void) {
      vTexCoord0 = ciTexCoord0;
      gl_Position = ciModelViewProjection * ciPosition;
   }
  ),
  // Fragment shader
  CI_GLSL(150,
   uniform sampler2D uTex0;
   in vec2 vTexCoord0;
   out vec4 oColor;

   void main(void) {
      oColor = texture(uTex0, vTexCoord0);
   }
  )
);
```



Visualizing UV Coordinates

```
auto shaders = gl::GlslProg::create(
  // Vertex shader
  CI_GLSL(150,
   uniform mat4 ciModelViewProjection;
   in vec4 ciPosition;
   in vec2 ciTexCoord0;
   out vec2 vTexCoord0;

   void main(void) {
      vTexCoord0 = ciTexCoord0;
      gl_Position = ciModelViewProjection * ciPosition;
   }
  ),
  // Fragment shader
  CI_GLSL(150,
   uniform sampler2D uTex0;
   in vec2 vTexCoord0;
   out vec4 oColor;

   void main(void) {
      oColor = vec4(vTexCoord0, 0, 1);
   }
  )
);
```

Look Familiar?

ciPosition

ciTexCoord0

Two Textures

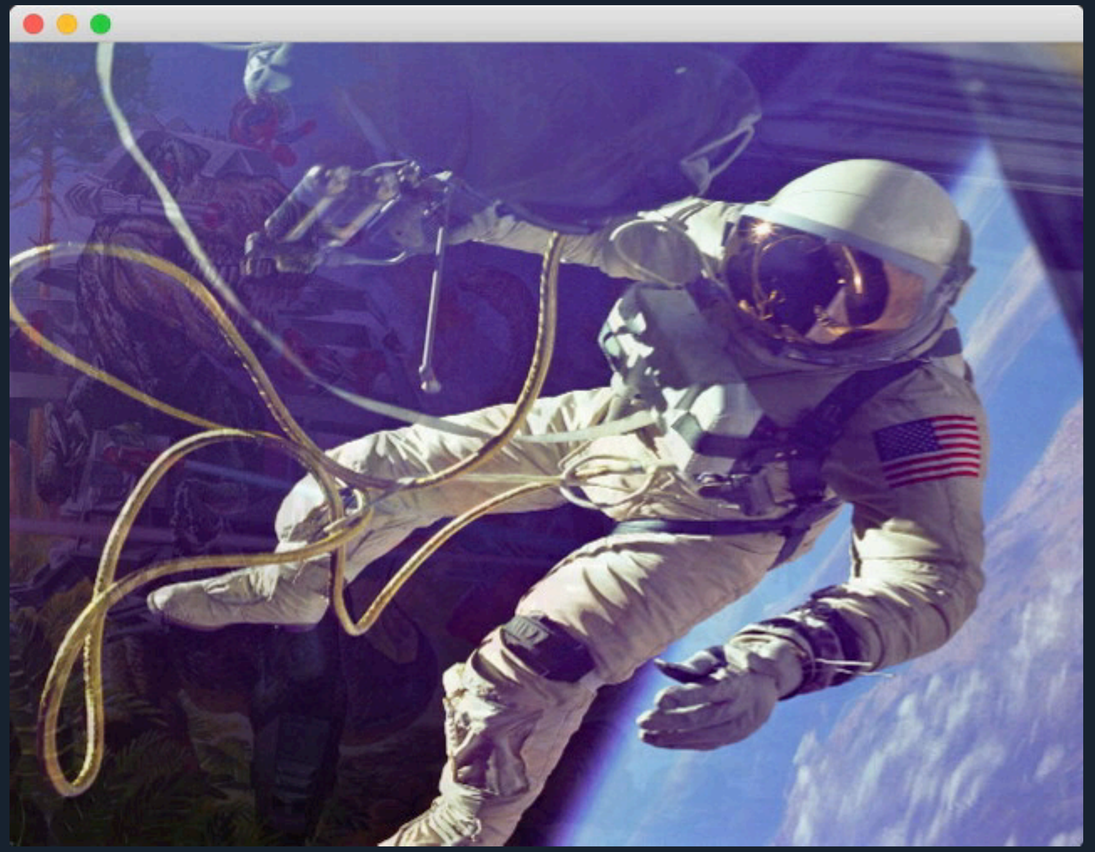
```
auto shaders = gl::GlslProg::create(
  // Vertex shader
  // ...
  // Fragment shader
  CI_GLSL(150,
   uniform sampler2D uTex0;
   uniform sampler2D uTex1;
   in vec2 vTexCoord0;
   out vec4 oColor;

   void main(void) {
      if (vTexCoord0.x < 0.5) {
         oColor = texture(uTex0, vTexCoord0);
      } else {
         oColor = texture(uTex1, vTexCoord0);
      }
   }
  )
);

mBatch = gl::Batch::create(geometry, shaders);

mTextureA = gl::Texture2d::create(loadImage(loadAsset("texture01.jpg")));
mTextureB = gl::Texture2d::create(loadImage(loadAsset("texture02.jpg")));

mTextureA->bind(0);
mTextureB->bind(1);

mBatch->getGlslProg()->uniform("uTex0", 0); // defined as 0 by default in Cinder
mBatch->getGlslProg()->uniform("uTex1", 1); // not defined by default in Cinder
```

Blending Two Textures

```
// Fragment shader
CI_GLSL(150,
  uniform sampler2D uTex0;
  uniform sampler2D uTex1;
  in vec2 vTexCoord0;
  out vec4 oColor;

  void main(void) {
   vec4 colorA = texture(uTex0, vTexCoord0);
   vec4 colorB = texture(uTex1, vTexCoord0);
   oColor = mix(colorA, colorB, vTexCoord0.x);
  }
)
```

Blending Two Textures

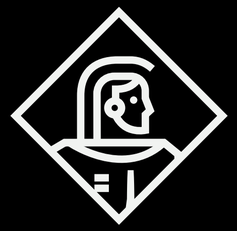
```
// Fragment shader
CI_GLSL(150,
  uniform sampler2D uTex0;
  uniform sampler2D uTex1;
  in vec2 vTexCoord0;
  out vec4 oColor;

  void main(void) {
   vec4 colorA = texture(uTex0, vTexCoord0);
   vec4 colorB = texture(uTex1, vTexCoord0);
   oColor = mix(colorA, colorB, 0.5 + 0.5 *
                sin(ciElapsedSeconds + vTexCoord0.x));
  }
)
```



Appendix

RESOURCES

1. TheBookOfShaders – Patricio Gonzales Vivo
    1. Book: http://thebookofshaders.com
    2. Editor: http://editor.thebookofshaders.com/
2. Cinder
    1. OpenGL Guide: https://www.libcinder.org/docs/guides/opengl/index.html
    2. Built-In GLSL Variables: https://github.com/cinder/Cinder/blob/e0056482c6bc19f1ba1937cd8ae321fc45ec08e6/src/cinder/gl/GlslProg.cpp#L616-L659
3. Shaderific
    1. GLSL functions: http://www.shaderific.com/glsl-functions
4. Nvidia GPU Gems (Advanced Shader Techniques): https://developer.nvidia.com/gpugems



BLUECADET