Advances in OpenGL ES

Session 505

Dan Omachi
GPU Software

Introduction

- OpenGL ES offers the most direct access to graphics hardware on iOS
  - Allows lots of flexibility and power
    - Flexibility can be challenging to master
- Utilizing the API to its fullest can set your app apart
  - Make the difference between shipping something good or great

Agenda

What you will learn

- New in iOS 7
  - Instancing
  - Vertex texture sampling
  - sRGB texture formats
- Tuning and performance
  - Understand the GPU pipeline
  - Insight into feedback from GPU tools

A Note About Power Efficiency

Rendering, but not…

Rendering Requires Power

- All GPUs used by iOS are power efficient
  - Still require considerable power to render

Rendering Requires Power

- All GPUs used by iOS are power efficient
  - Still require considerable power to render
- Manage frame rate
  - Use `CADisplayLink` to sync to display
  - Target 30 fps

Rendering Requires Power

- All GPUs used by iOS are power efficient
  - Still require considerable power to render
- Manage frame rate
  - Use `CADisplayLink` to sync to display
  - Target 30 fps
- Often unnecessary to render at all
  - Do not re-render a frame if it looks the same as the previous frame
    - No animation or movement

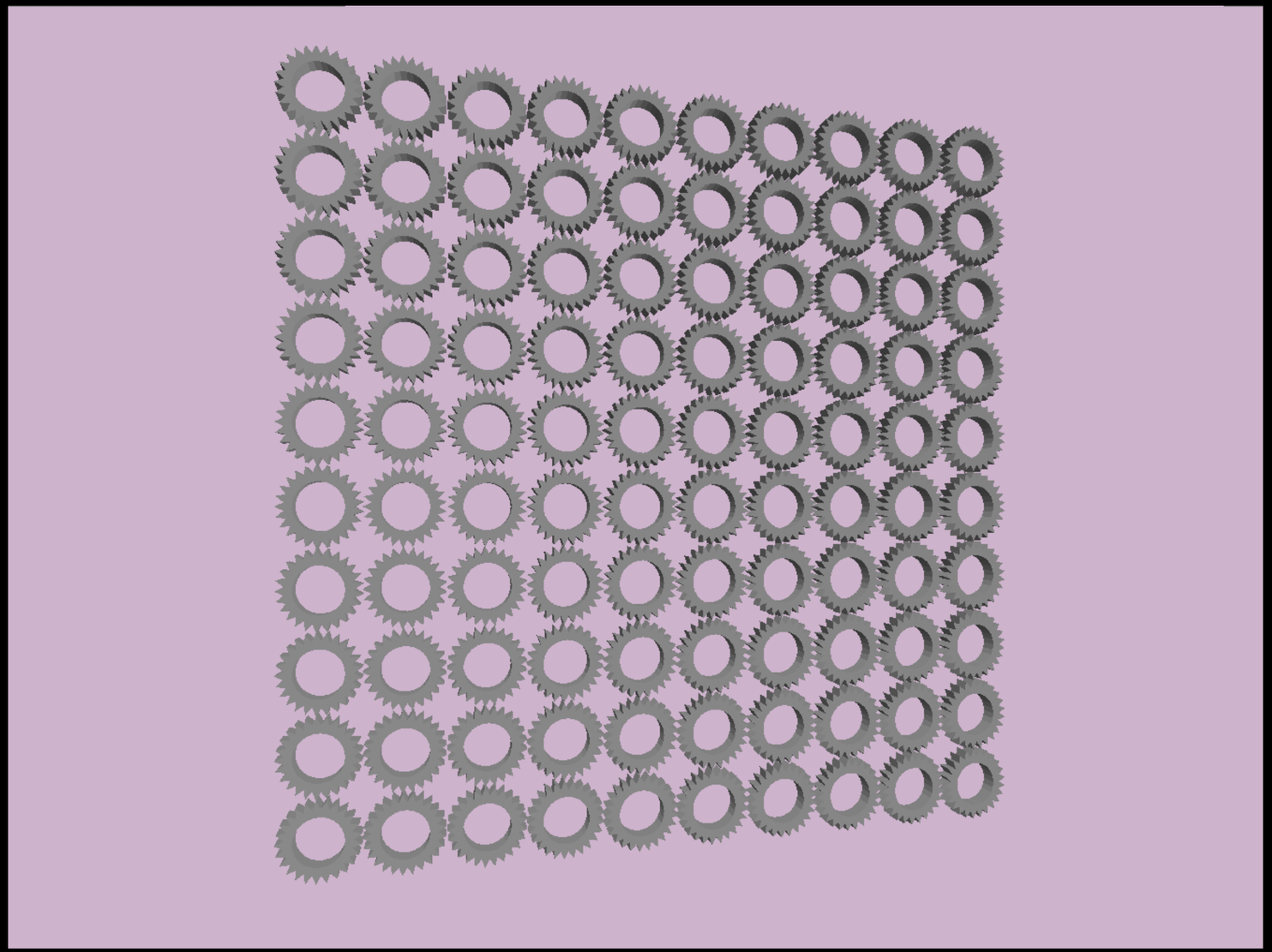
Rendering Requires Power

- All GPUs used by iOS are power efficient
  - Still require considerable power to render
- Manage frame rate
  - Use `CADisplayLink` to sync to display
  - Target 30 fps
- Often unnecessary to render at all
  - Do not re-render a frame if it looks the same as the previous frame
    - No animation or movement
- Particularly important with the multilayered iOS 7 UI
  - UI can skip compositing if nothing has changed in a layer

Instancing

From one, come many…

Drawing Many Models



Drawing Many Models

Without instancing

```
for(x = 0; x < 10; x++)
{
    for(y = 0; y < 10; y++)
    {
      // Set uniform to position the gear
      glUniform4fv(mygearPosition[x][y]);

      // Draw the gear
      glDrawArrays(GL_TRIANGLES, 0, numGearVertices);
    }
}
```

Drawing Many Models

Without instancing

```
for(x = 0; x < 10; x++)
{
    for(y = 0; y < 10; y++)
    {
        // Set uniform to position the gear
        glUniform4fv(mygearPosition[x][y]);

        // Draw the gear
        glDrawArrays(GL_TRIANGLES, 0, numGearVertices);
    }
}
```

Drawing Many Models

Without instancing

```
for(x = 0; x < 10; x++)
{
    for(y = 0; y < 10; y++)
    {
        // Set uniform to position the gear
        glUniform4fv(mygearPosition[x][y]);

        // Draw the gear
        glDrawArrays(GL_TRIANGLES, 0, numGearVertices);
    }
}
```

Drawing Many Models

Without instancing

```
for(x = 0; x < 10; x++)
{
    for(y = 0; y < 10; y++)
    {
      // Set uniform to position the gear
      glUniform4fv(mygearPosition[x][y]);
      // Draw the gear
      glDrawArrays(GL_TRIANGLES, 0, numGearVertices);
    }
}
```

Drawing Many Models

Without instancing

```
for(x = 0; x < 10; x++)
{
    for(y = 0; y < 10; y++)
    {
      // Set uniform to position the gear
      glUniform4fv(mygearPosition[x][y]);

      // Draw the gear
      glDrawArrays(GL_TRIANGLES, 0, numGearVertices);
    }
}
```

What Instancing Does

- Draw the same model many times
  - Different parameter for each instance
  - Single draw call
- Each instance can have different:
  - Position
  - Matrices
  - Texture coordinates

Instancing

Two forms

- Instanced arrays
  - `GL_APPLE_instanced_arrays`
  - Instance parameters in a vertex array
- Shader instance ID
  - `GL_APPLE_draw_instanced`
  - ID variable for instance drawn in vertex shader

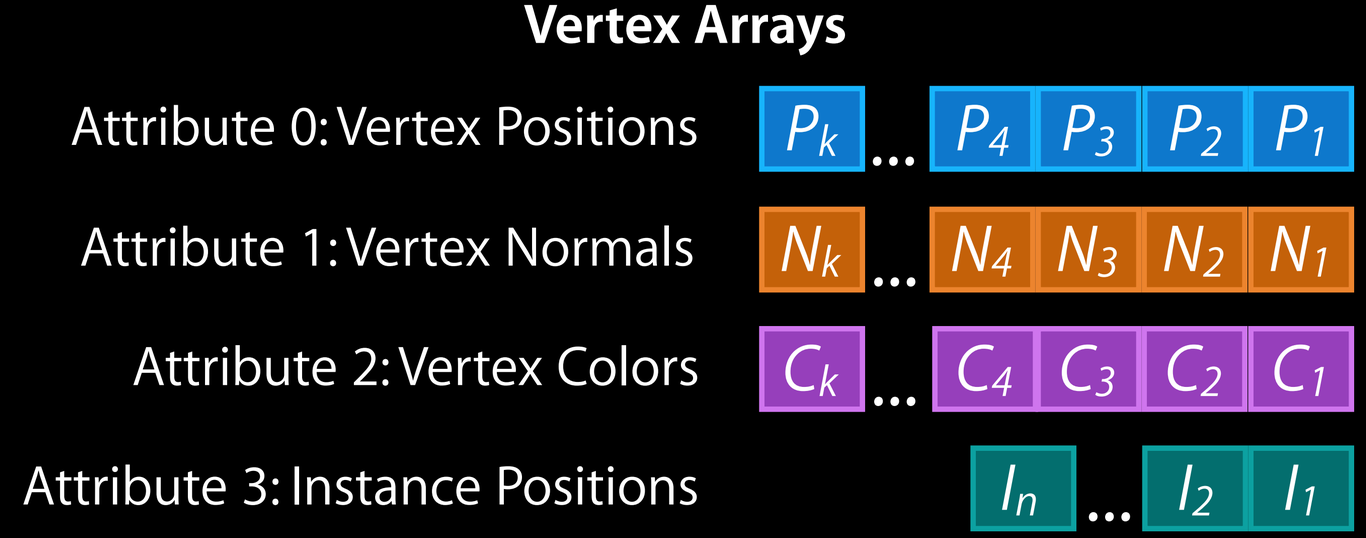
Instancing

Method 1: Instanced arrays

- `glVertexAttribDivisorAPPLE` indicates
  - Attribute array supplying instance data
  - Number of instances to draw before advancing to the next element in this array
- Draw with `glDrawArraysInstancedAPPLE` or `glDrawElementsInstancedAPPLE`
  - Same as `glDrawArrays` and `glDrawElements`, but extra parameter indicates number of instances to draw

Instancing

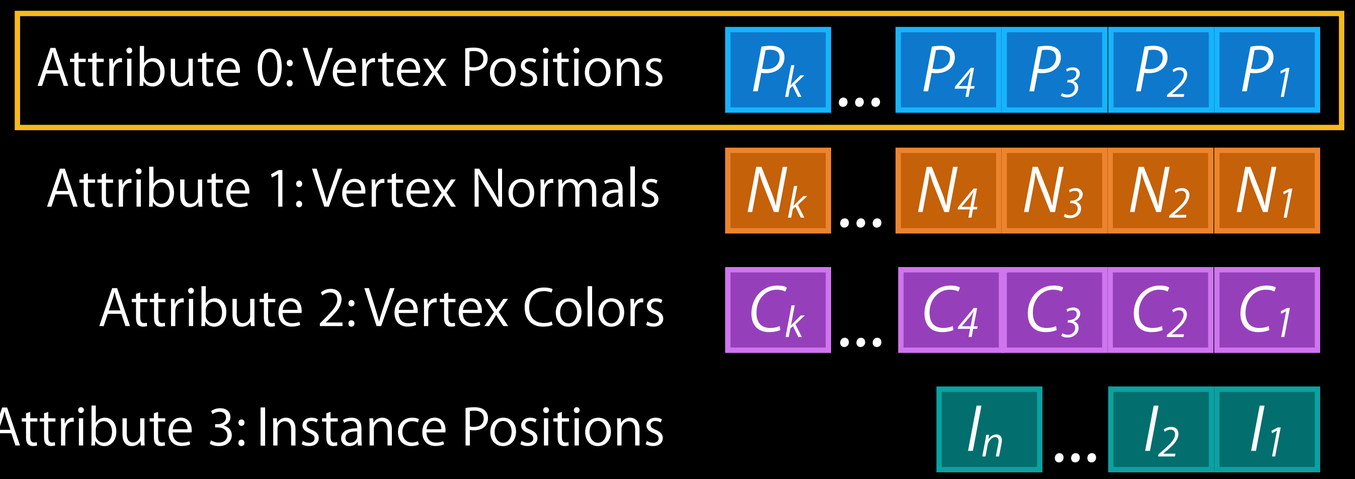
Method 1: Instanced arrays



Instancing

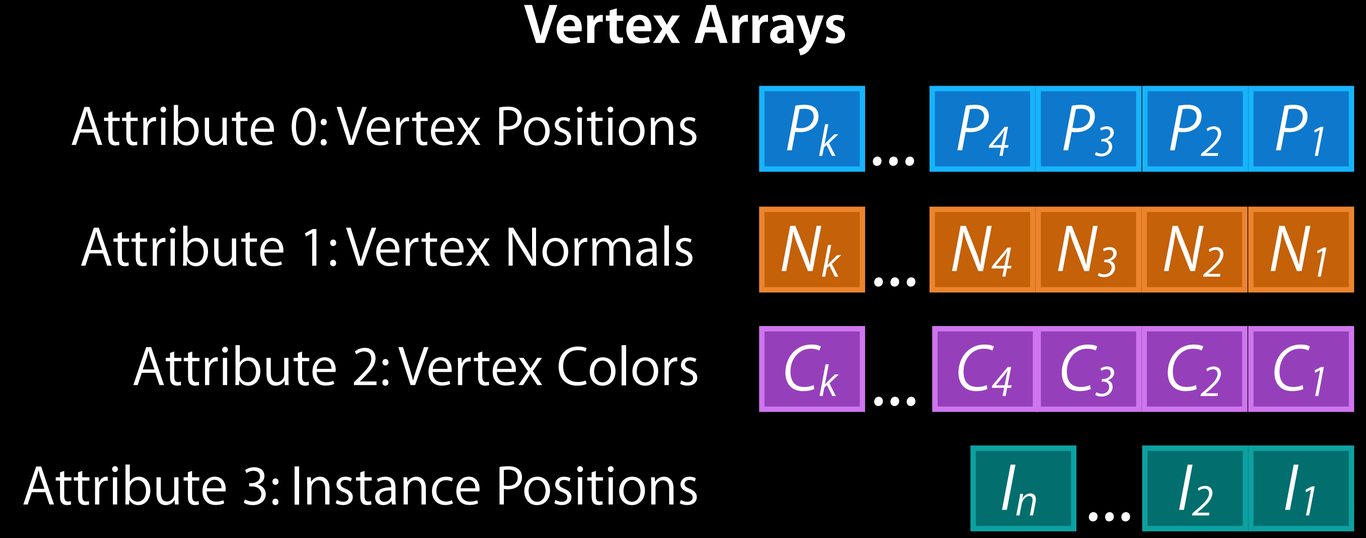
Method 1: Instanced arrays

```
glVertexAttribPointer(0, ... , positionOffset);
```



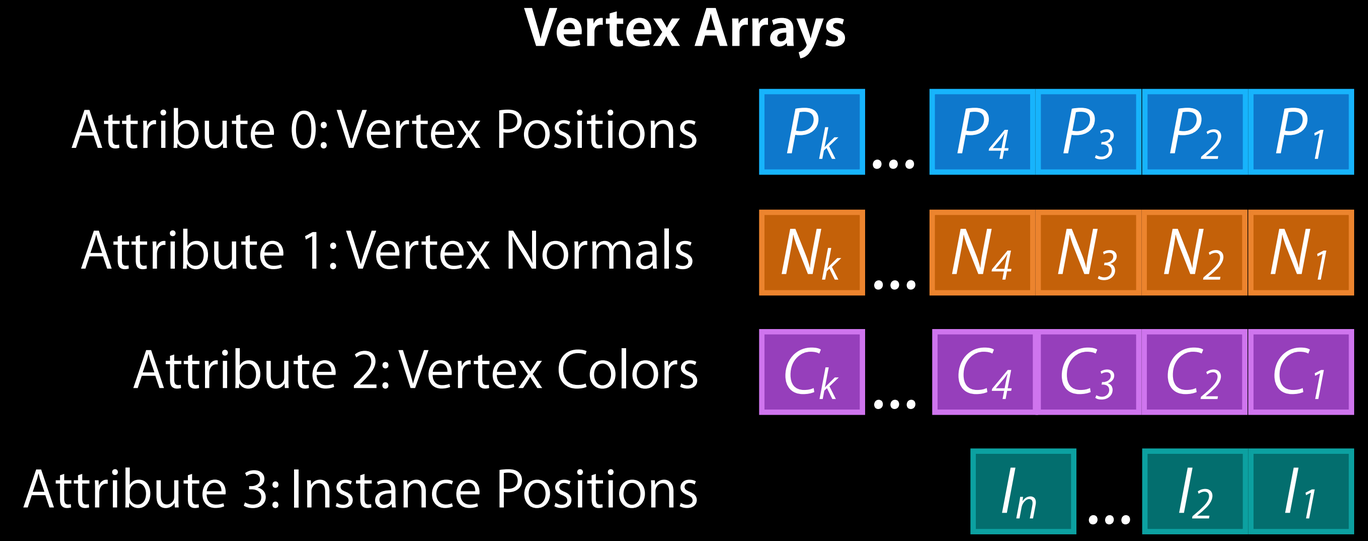
Instancing

Method 1: Instanced arrays



Instancing

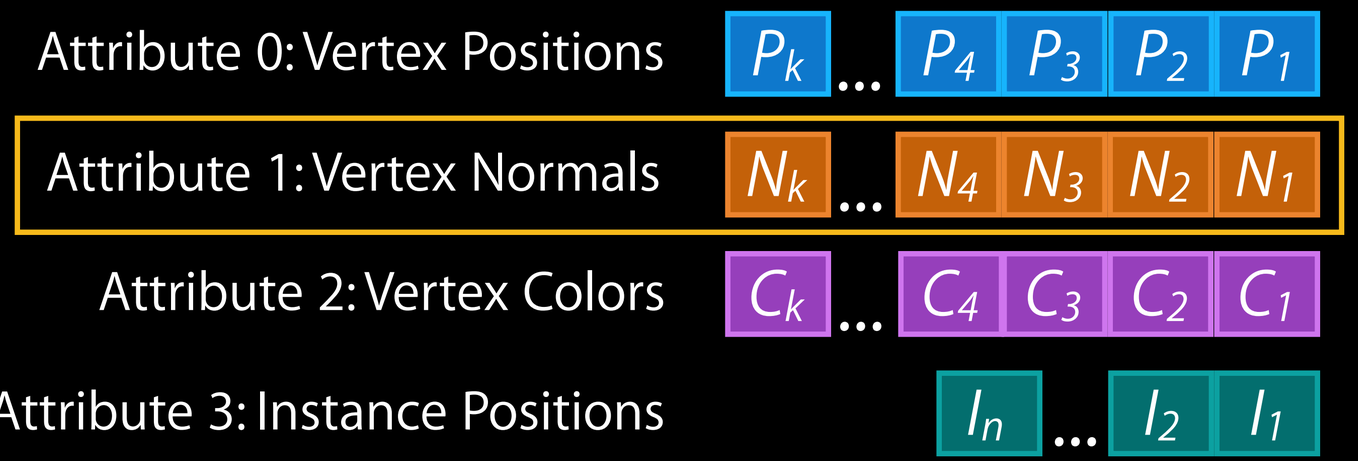
Method 1: Instanced arrays



Instancing

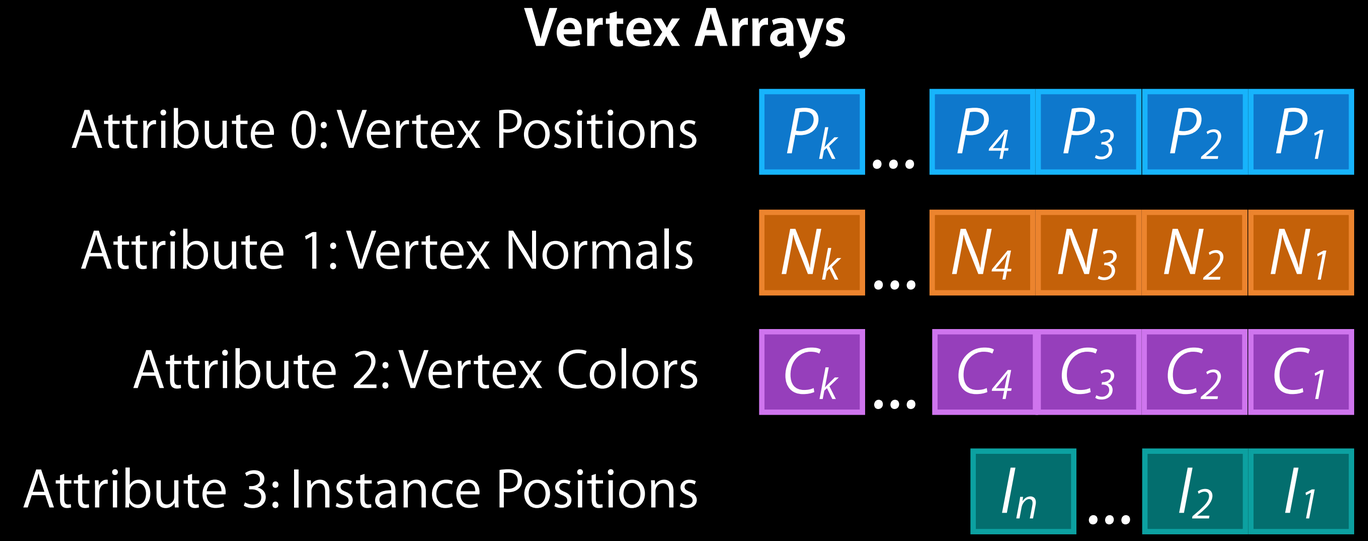
Method 1: Instanced arrays

```
glVertexAttribPointer(1, ... , normalOffset);
```



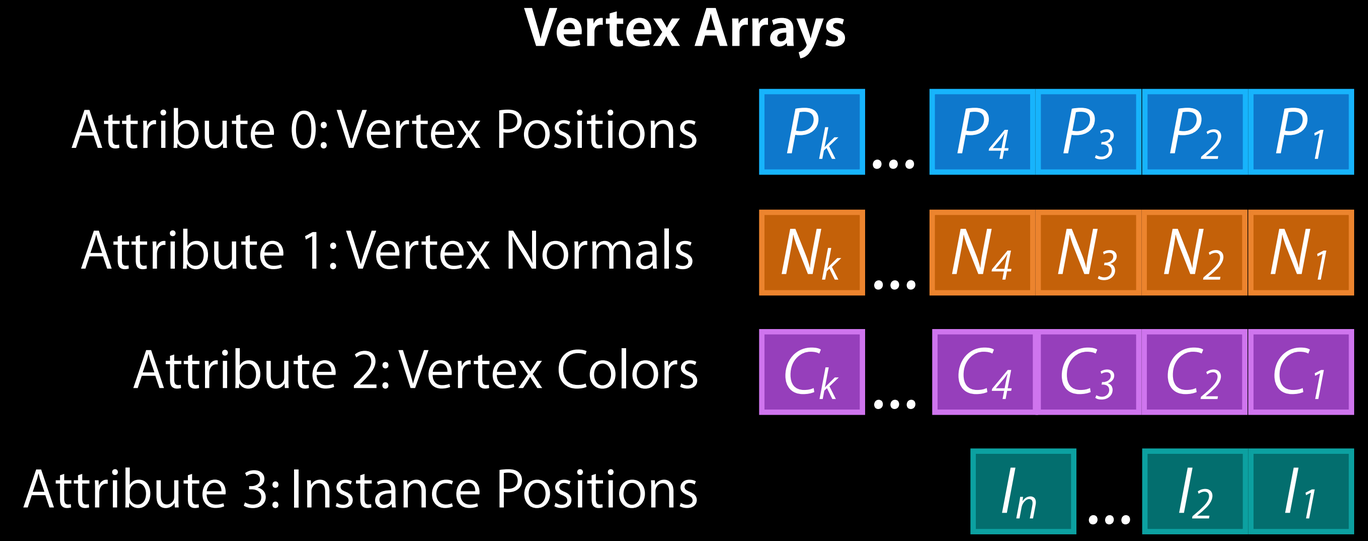
Instancing

Method 1: Instanced arrays



Instancing

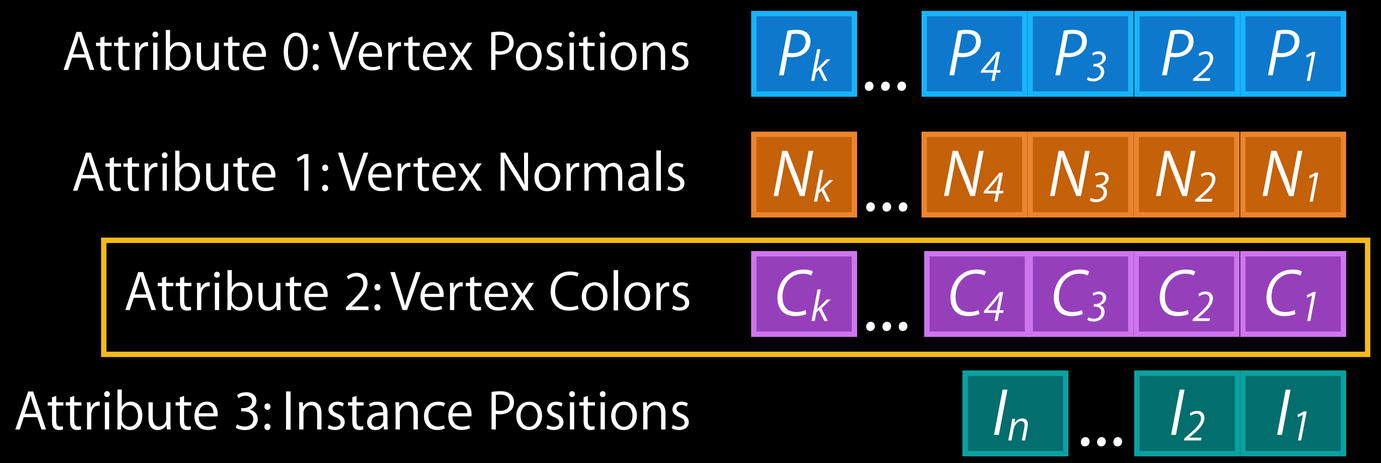
Method 1: Instanced arrays



Instancing

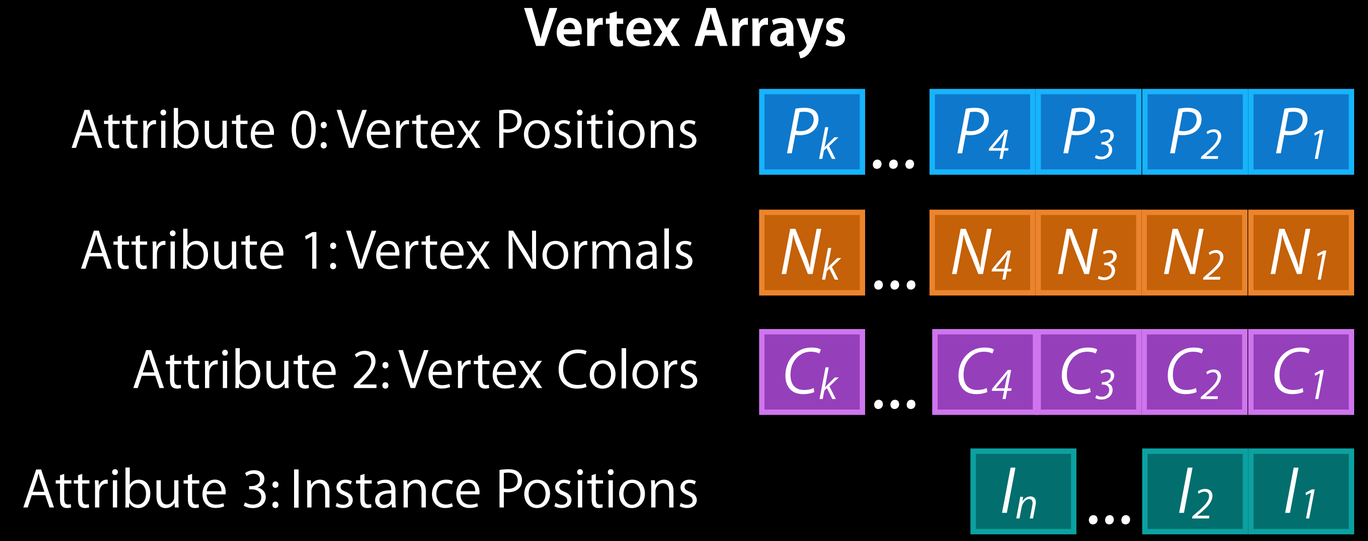
Method 1: Instanced arrays

```
glVertexAttribPointer(2, ... , colorOffset);
```



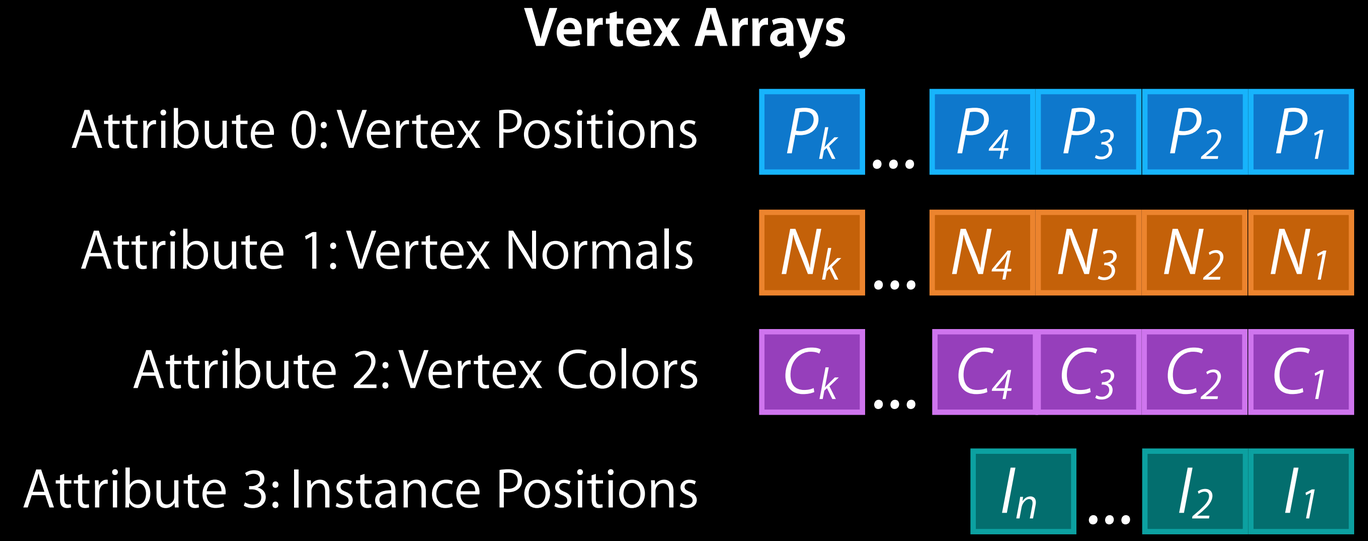
Instancing

Method 1: Instanced arrays



Instancing

Method 1: Instanced arrays

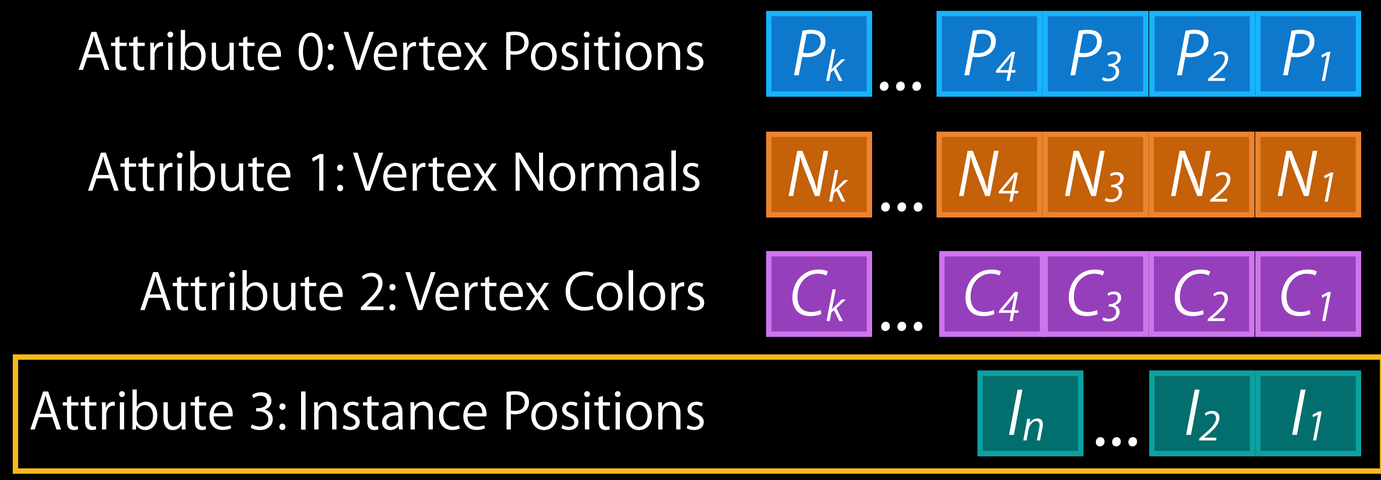


Instancing

Method 1: Instanced arrays

```
glVertexAttribPointer(3, ..., instancePositionOffset);
```

Vertex Arrays

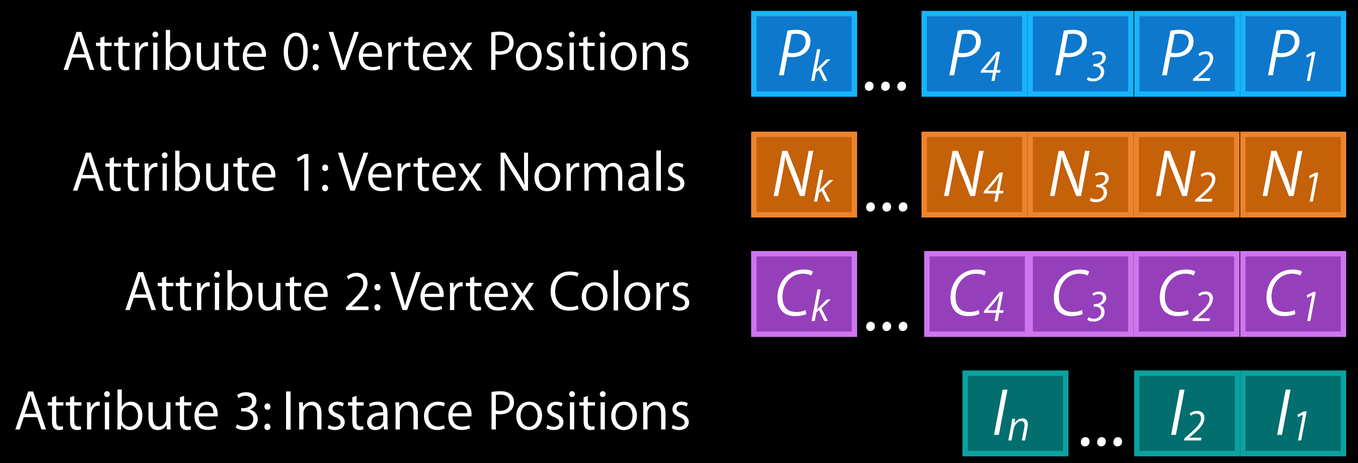


Instancing

Method 1: Instanced arrays

```
glVertexAttribPointer(3, ..., instancePositionOffset);
```

Vertex Arrays



Instancing

Method 1: Instanced arrays

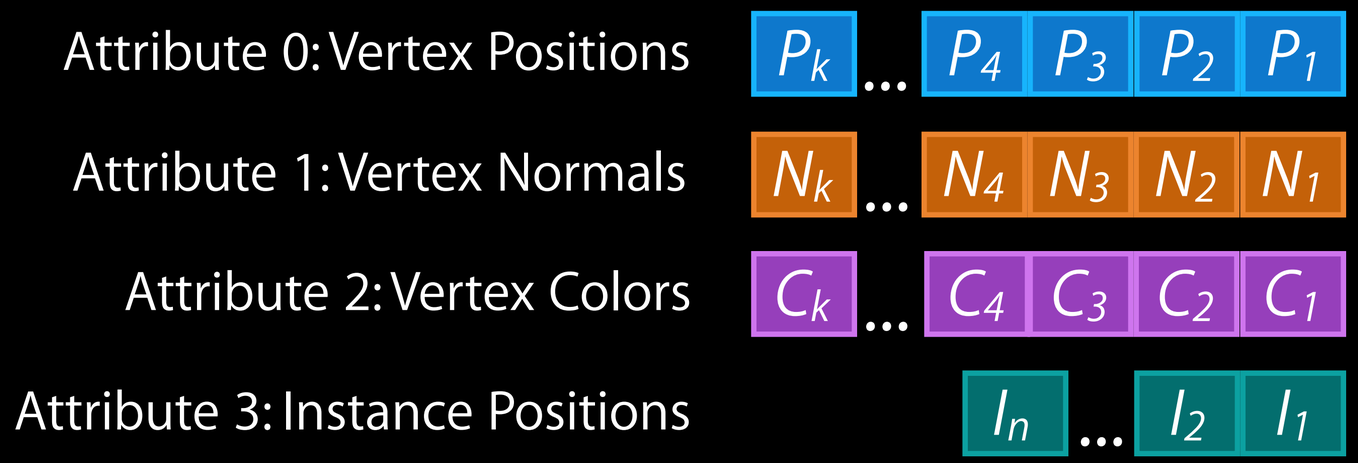
```
glVertexAttribPointer(3, ..., instancePositionOffset);
```

Instancing

Method 1: Instanced arrays

```
glVertexAttribPointer(3, ..., instancePositionOffset);

glVertexAttribDivisorAPPLE(3, 1);
```

Vertex Arrays



Instancing

Method 1: Instanced arrays

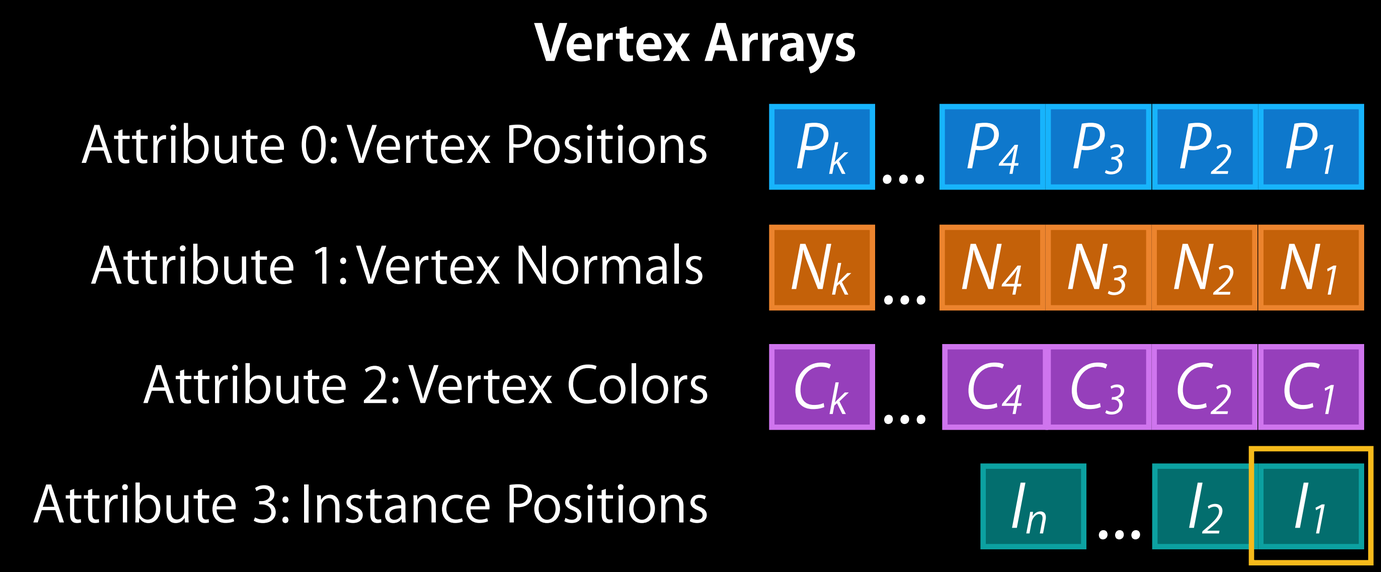
```
glVertexAttribPointer(3, ..., instancePositionOffset);

glVertexAttribDivisorAPPLE(3, 1);
```

Vertex Arrays

Instancing

Method 1: Instanced arrays

```
glVertexAttribPointer(3, ..., instancePositionOffset);

glVertexAttribDivisorAPPLE(3, 1);
```

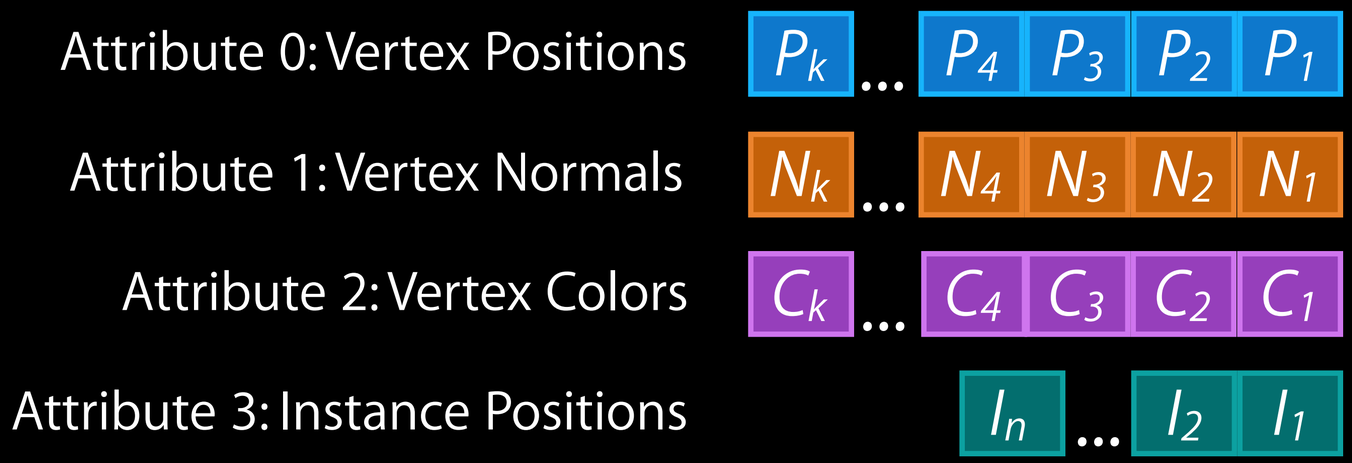


Instancing

Method 1: Instanced arrays

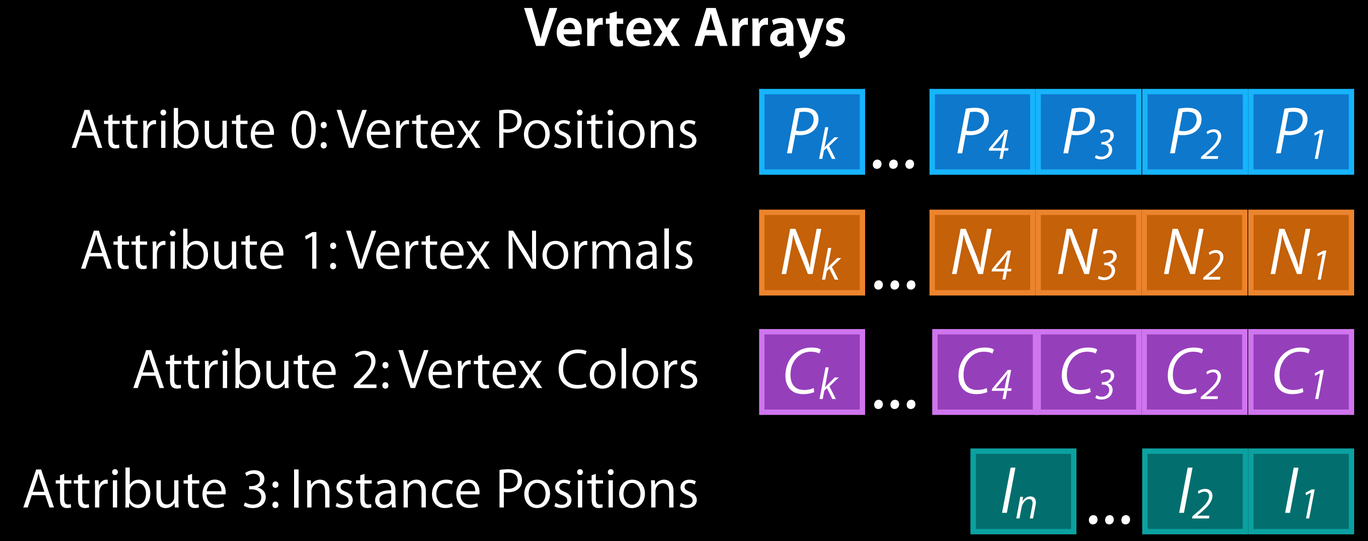
```
glVertexAttribPointer(3, ..., instancePositionOffset);

glVertexAttribDivisorAPPLE(3, 1);
```

Vertex Arrays



Instancing

Method 1: Instanced arrays



Instancing

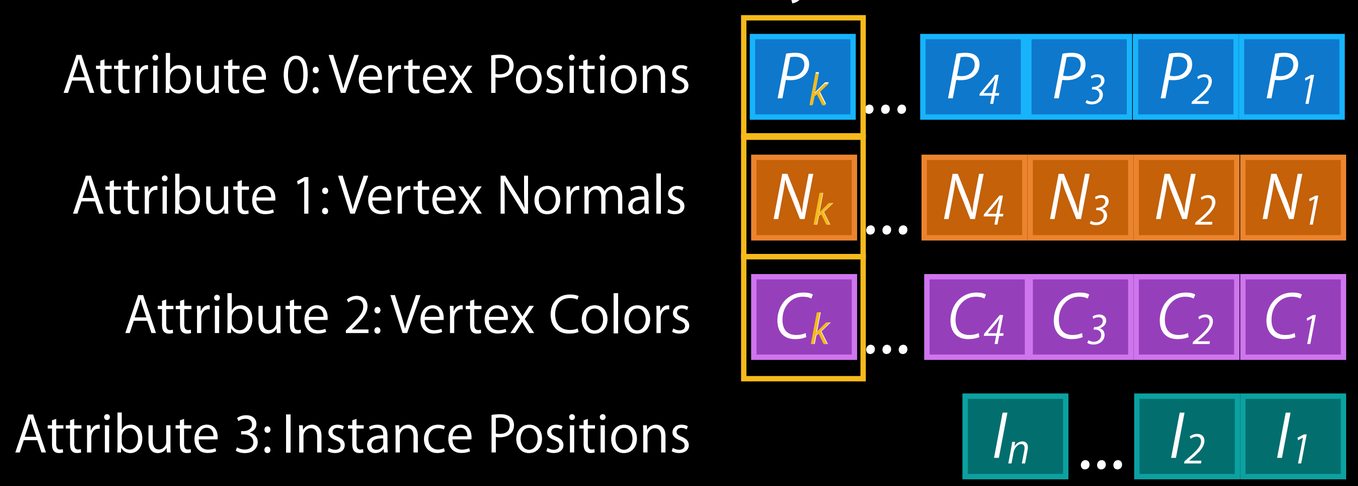
Method 1: Instanced arrays

```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```

Instancing

Method 1: Instanced arrays

```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```



Instancing

Method 1: Instanced arrays

```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```

Instancing

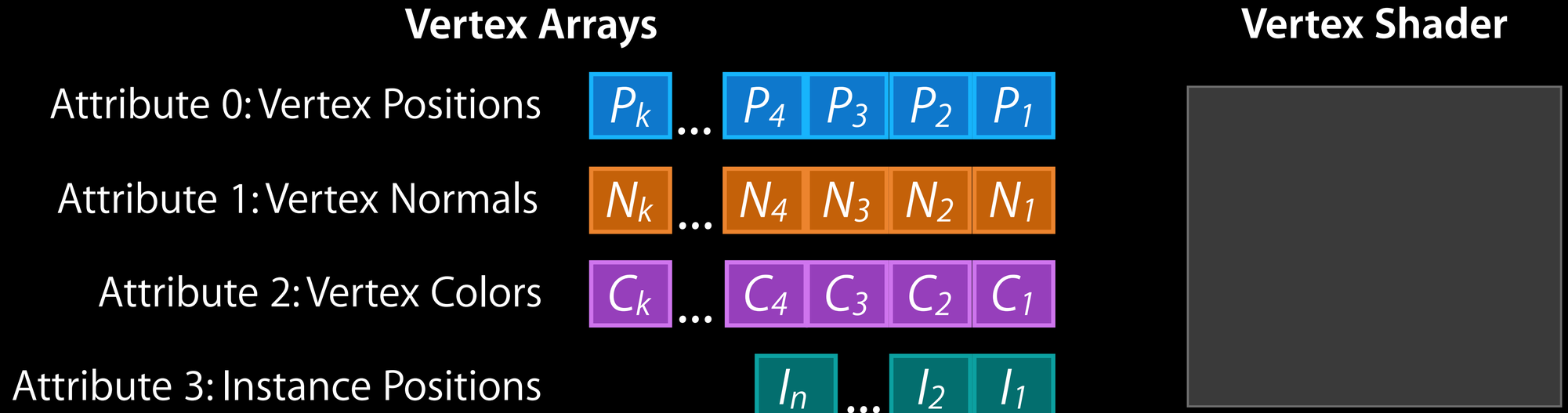
Method 1: Instanced arrays

```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```

Instancing

Method 1: Instanced arrays

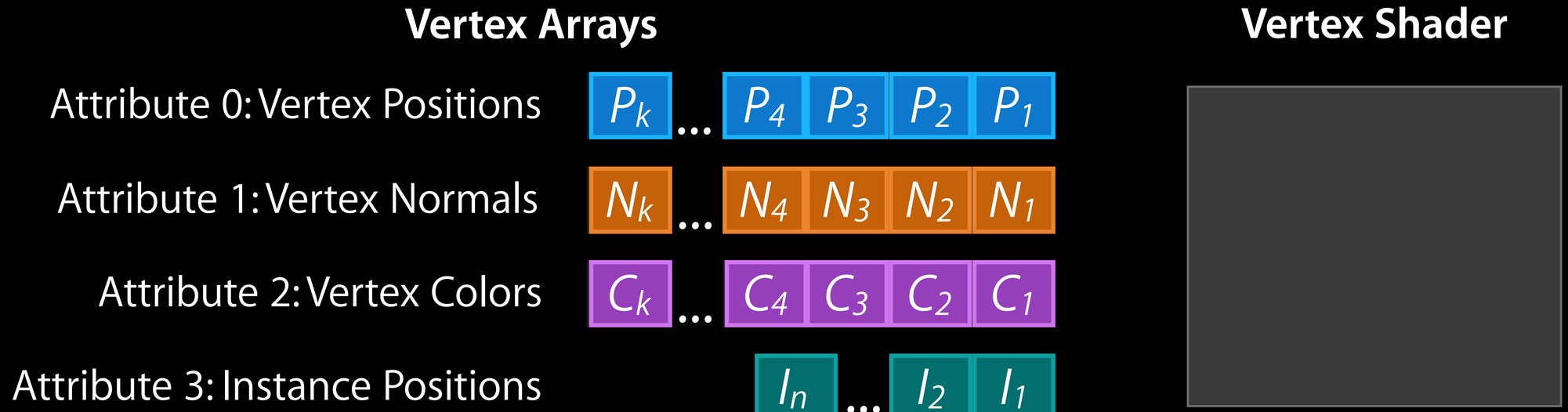
```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```



Instancing

Method 1: Instanced arrays

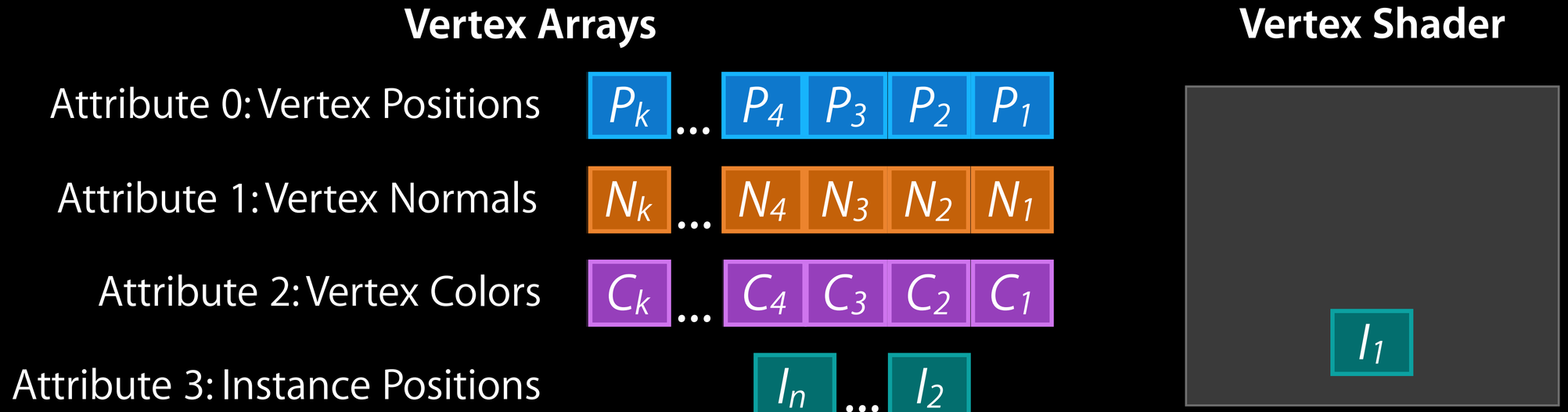
```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```



Instancing

Method 1: Instanced arrays

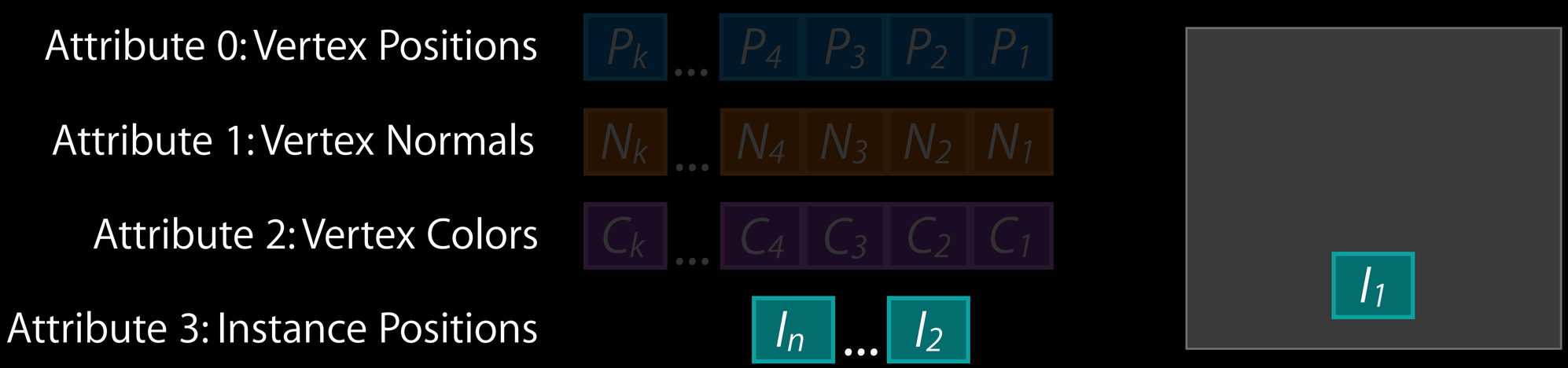
```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```



Instancing

Method 1: Instanced arrays

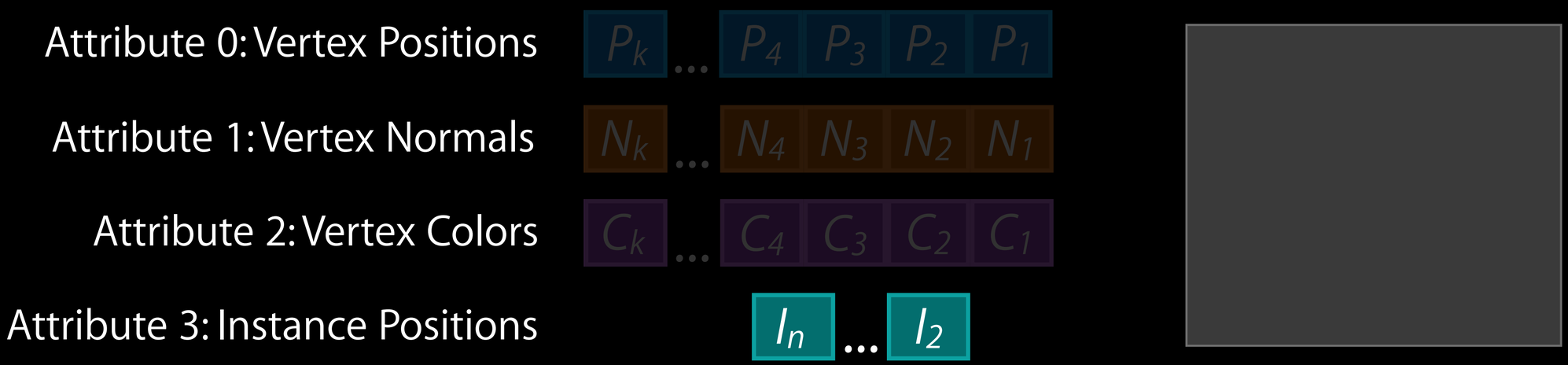
```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```



Instancing

Method 1: Instanced arrays

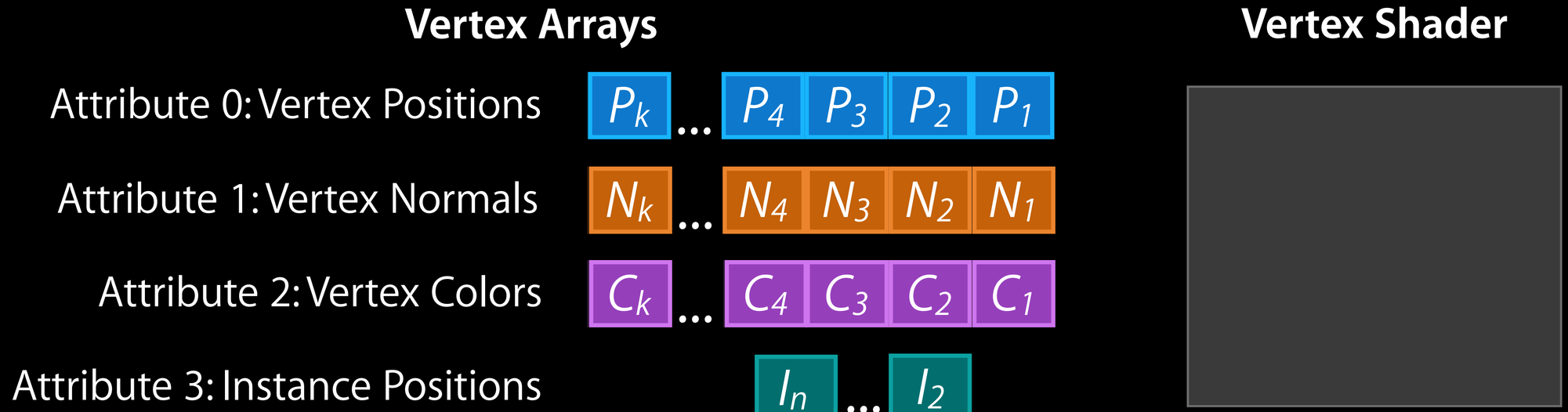
```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```



Instancing

Method 1: Instanced arrays

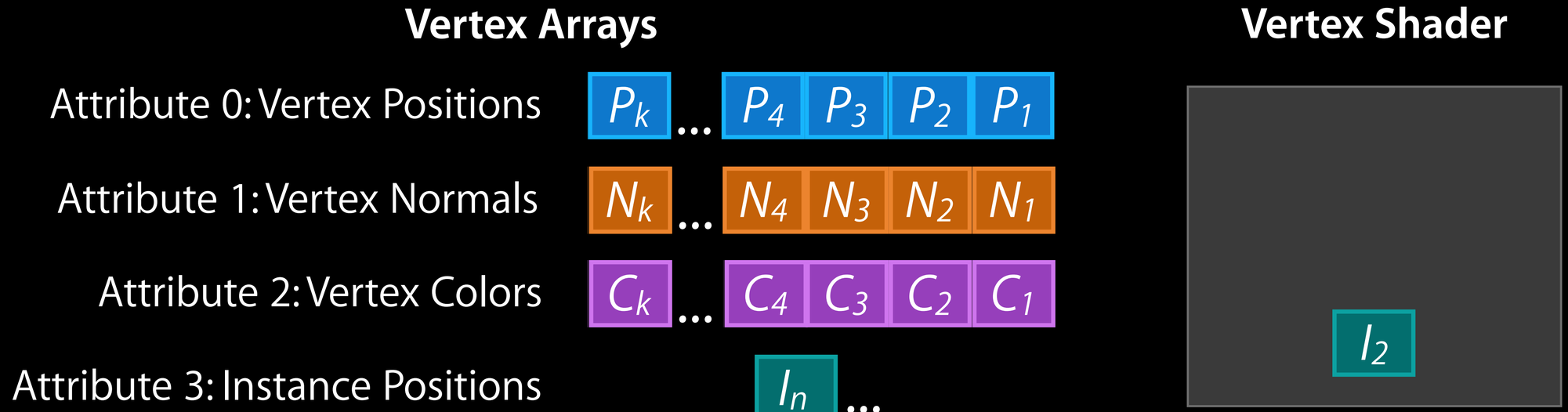
```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```



Instancing

Method 1: Instanced arrays

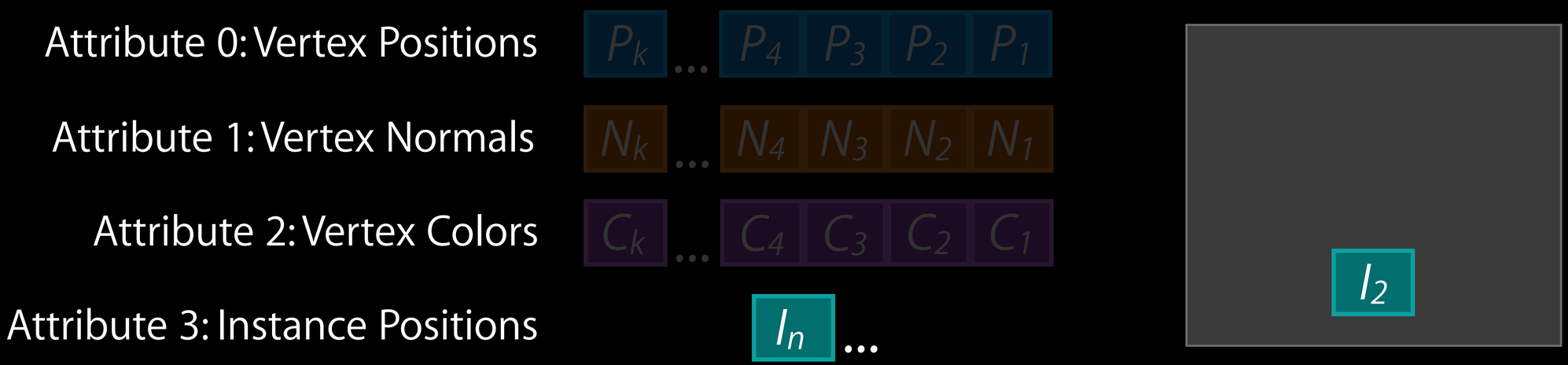
```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```



Instancing

Method 1: Instanced arrays

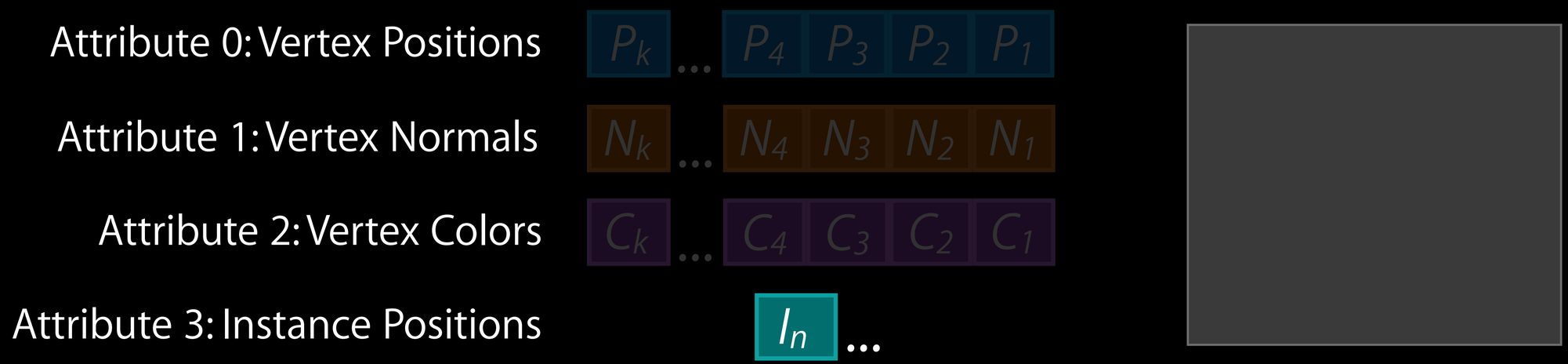
```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```



Instancing

Method 1: Instanced arrays

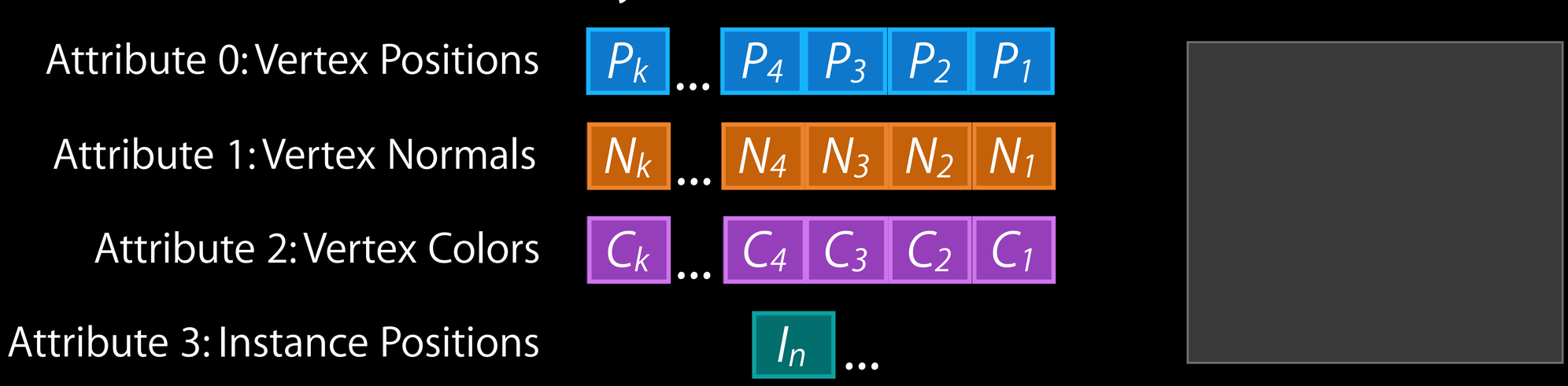
```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```



Instancing

Method 1: Instanced arrays

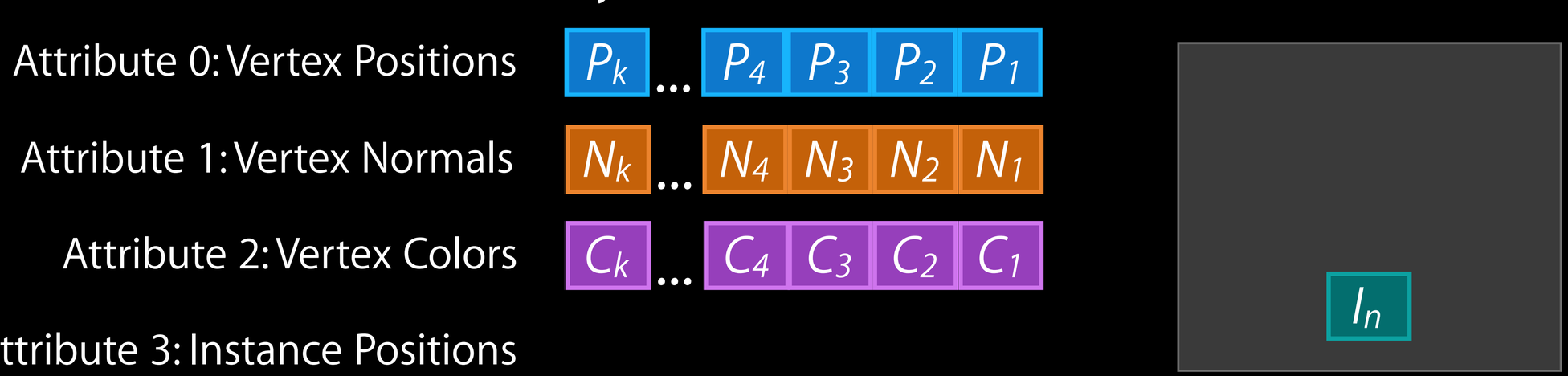
```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```



Instancing

Method 1: Instanced arrays

```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```



Instancing

Method 1: Instanced arrays

```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```

Vertex Arrays

Attribute 0: Vertex Positions

Attribute 1: Vertex Normals

Attribute 2: Vertex Colors

Attribute 3: Instance Positions

Vertex Shader

Instancing

Method 1: Instanced arrays

```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```

Vertex Arrays

Attribute 0: Vertex Positions

Attribute 1: Vertex Normals

Attribute 2: Vertex Colors

Attribute 3: Instance Positions

Vertex Shader

Instanced Arrays

API setup


```
// Setup per vertex position, normal, and color arrays on
//  indices 0, 1, and 2 for gear model same as usually

...

// Setup vertex array index 3 to contain per object positions
glVertexAttribPointer(3, ..., instancePositionOffset);

// Indicate that attribute 3 should increment 1 element each instance
glVertexAttribDivisorAPPLE(3, 1);

// Draw gear model 100 times
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, NUM_GEAR_VERTS, 100);
```

Instanced Arrays

API setup


```
// Setup per vertex position, normal, and color arrays on
//  indices 0, 1, and 2 for gear model same as usually

...
```

```
// Setup vertex array index 3 to contain per object positions
glVertexAttribPointer(3, ..., instancePositionOffset);

// Indicate that attribute 3 should increment 1 element each instance
glVertexAttribDivisorAPPLE(3, 1);

// Draw gear model 100 times
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, NUM_GEAR_VERTS, 100);
```

Instanced Arrays

API setup


```
// Setup per vertex position, normal, and color arrays on
//  indices 0, 1, and 2 for gear model same as usually

...

// Setup vertex array index 3 to contain per object positions
glVertexAttribPointer(3, ..., instancePositionOffset);

// Indicate that attribute 3 should increment 1 element each instance
glVertexAttribDivisorAPPLE(3, 1);

// Draw gear model 100 times
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, NUM_GEAR_VERTS, 100);
```

Instanced Arrays

API setup


```
// Setup per vertex position, normal, and color arrays on
//  indices 0, 1, and 2 for gear model same as usually

...

// Setup vertex array index 3 to contain per object positions
glVertexAttribPointer(3, ..., instancePositionOffset);

// Indicate that attribute 3 should increment 1 element each instance
glVertexAttribDivisorAPPLE(3, 1);

// Draw gear model 100 times
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, NUM_GEAR_VERTS, 100);
```

Instanced Arrays

API setup


```
// Setup per vertex position, normal, and color arrays on
//  indices 0, 1, and 2 for gear model same as usually

...

// Setup vertex array index 3 to contain per object positions
glVertexAttribPointer(3, ..., instancePositionOffset);

// Indicate that attribute 3 should increment 1 element each instance
glVertexAttribDivisorAPPLE(3, 1);

// Draw gear model 100 times
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, NUM_GEAR_VERTS, 100);
```

Instanced Arrays

API setup


```
// Setup per vertex position, normal, and color arrays on
//  indices 0, 1, and 2 for gear model same as usually

...

// Setup vertex array index 3 to contain per object positions
glVertexAttribPointer(3, ..., instancePositionOffset);

// Indicate that attribute 3 should increment 1 element each instance
glVertexAttribDivisorAPPLE(3, 1);

// Draw gear model 100 times
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, NUM_GEAR_VERTS, 100);
```

Instance Arrays

Vertex shader

```
// Per vertex and normal position
attribute vec4 pos;
attribute vec3 normal;
// Per instance position (maps to attribute index 3)
attribute vec4 instancePos;
...
void main()
{
    // Add vertex position to instance position
    vec4 tempPos = instancePos + pos;
    // Transform position to clip space and output to gl_Position

    ...

    gl_Position = tempPos;
}
```

Instance Arrays

Vertex shader

```
// Per vertex and normal position
attribute vec4 pos;
attribute vec3 normal;
// Per instance position (maps to attribute index 3)
attribute vec4 instancePos;
...
void main()
{
    // Add vertex position to instance position
    vec4 tempPos = instancePos + pos;
    // Transform position to clip space and output to gl_Position

    ...

    gl_Position = tempPos;
}
```

Instance Arrays

Vertex shader

```
// Per vertex and normal position
attribute vec4 pos;
attribute vec3 normal;
// Per instance position (maps to attribute index 3)
attribute vec4 instancePos;
...
void main()
{
    // Add vertex position to instance position
    vec4 tempPos = instancePos + pos;
    // Transform position to clip space and output to gl_Position

    ...

    gl_Position = tempPos;
}
```

Instance Arrays

Vertex shader

```
// Per vertex and normal position
attribute vec4 pos;
attribute vec3 normal;
// Per instance position (maps to attribute index 3)
attribute vec4 instancePos;
...
void main()
{
    // Add vertex position to instance position
    vec4 tempPos = instancePos + pos;
    // Transform position to clip space and output to gl_Position

    ...

    gl_Position = tempPos;
}
```

Instance Arrays

Vertex shader

```
// Per vertex and normal position
attribute vec4 pos;
attribute vec3 normal;
// Per instance position (maps to attribute index 3)
attribute vec4 instancePos;
...

void main()
{
    // Add vertex position to instance position
    vec4 tempPos = instancePos + pos;
    // Transform position to clip space and output to gl_Position

    ...

    gl_Position = tempPos;
}
```

Instance Arrays

Vertex shader

```
// Per vertex and normal position
attribute vec4 pos;
attribute vec3 normal;
// Per instance position (maps to attribute index 3)
attribute vec4 instancePos;
...
void main()
{
    // Add vertex position to instance position
    vec4 tempPos = instancePos + pos;
    // Transform position to clip space and output to gl_Position

    ...

    gl_Position = tempPos;
}
```

Instancing

Method 2: Instance ID

- Built-in `gl_InstanceIDAPPLE` variable in vertex shader
- ID incremented for each instance
- Can use ID to calculate unique info for instance
  - Calculate texture or position
  - Location of instance parameter in uniform array or texture
- Also uses `glDrawArraysInstancedAPPLE` or `glDrawElementsInstancedAPPLE`

Instancing

Method 2: Instance ID

```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```

Vertex Arrays

Attribute 0: Vertex Positions — P_k ... P_4 P_3 P_2 P_1

Attribute 1: Vertex Normals — N_k ... N_4 N_3 N_2 N_1

Attribute 2: Vertex Colors — C_k ... C_4 C_3 C_2 C_1

Vertex Shader

```
gl_InstanceID
```

Instancing

Method 2: Instance ID

```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```

Vertex Arrays

Attribute 0: Vertex Positions P_k ... P_4 P_3 P_2 P_1

Attribute 1: Vertex Normals N_k ... N_4 N_3 N_2 N_1

Attribute 2: Vertex Colors C_k ... C_4 C_3 C_2 C_1

Vertex Shader

```
gl_InstanceID
0
```

Instancing

Method 2: Instance ID

```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```

Vertex Arrays

Attribute 0: Vertex Positions — P_k ... P_4 P_3 P_2 P_1

Attribute 1: Vertex Normals — N_k ... N_4 N_3 N_2 N_1

Attribute 2: Vertex Colors — C_k ... C_4 C_3 C_2 C_1

Vertex Shader

```
gl_InstanceID
```

Instancing

Method 2: Instance ID

```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```

Vertex Arrays

Attribute 0: Vertex Positions P_k ... P_4 P_3 P_2 P_1

Attribute 1: Vertex Normals N_k ... N_4 N_3 N_2 N_1

Attribute 2: Vertex Colors C_k ... C_4 C_3 C_2 C_1

Vertex Shader

```
gl_InstanceID
1
```

Instancing

Method 2: Instance ID

```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```

Vertex Arrays

Attribute 0: Vertex Positions — P_k ... P_4 P_3 P_2 P_1

Attribute 1: Vertex Normals — N_k ... N_4 N_3 N_2 N_1

Attribute 2: Vertex Colors — C_k ... C_4 C_3 C_2 C_1

Vertex Shader

```
gl_InstanceID
```

Instancing

Method 2: Instance ID

```
glDrawArraysInstancedAPPLE(GL_TRIANGLES, 0, k, n);
```

Vertex Arrays

Attribute 0: Vertex Positions

Attribute 1: Vertex Normals

Attribute 2: Vertex Colors

Vertex Shader

```
gl_InstanceID
```

n

Shader Instance ID

Vertex shader


```
uniform float gearSize;
attribute vec2 vertexPosition;

...

void main() {
    float instanceID = float(gl_InstanceIDAPPLE);
    instancePosition.x = mod(instanceID, 10) * gearSize;
    instancePosition.y = (floor(instanceID)/ 10) * gearSize;
    vec4 tempPos = vertexPosition.xy + instancePosition.xy;

    // Transform tempPos to clip space and output to gl_Position

    ...

    gl_Position = tempPos;
}
```

Shader Instance ID

Vertex shader


```
uniform float gearSize;
attribute vec2 vertexPosition;

...

void main() {
    float instanceID = float(gl_InstanceIDAPPLE);
    instancePosition.x = mod(instanceID, 10) * gearSize;
    instancePosition.y = (floor(instanceID)/ 10) * gearSize;
    vec4 tempPos = vertexPosition.xy + instancePosition.xy;

    // Transform tempPos to clip space and output to gl_Position

    ...

    gl_Position = tempPos;
}
```

Shader Instance ID

Vertex shader


```
uniform float gearSize;
attribute vec2 vertexPosition;

...

void main() {
    float instanceID = float(gl_InstanceIDAPPLE);
    instancePosition.x = mod(instanceID, 10) * gearSize;
    instancePosition.y = (floor(instanceID)/ 10) * gearSize;
    vec4 tempPos = vertexPosition.xy + instancePosition.xy;

    // Transform tempPos to clip space and output to gl_Position

    ...

    gl_Position = tempPos;
}
```

Shader Instance ID

Vertex shader


```
uniform float gearSize;
attribute vec2 vertexPosition;

...

void main() {
    float instanceID = float(gl_InstanceIDAPPLE);
    instancePosition.x = mod(instanceID, 10) * gearSize;
    instancePosition.y = (floor(instanceID)/ 10) * gearSize;
    vec4 tempPos = vertexPosition.xy + instancePosition.xy;

    // Transform tempPos to clip space and output to gl_Position

    ...

    gl_Position = tempPos;
}
```

Shader Instance ID

Vertex shader


```
uniform float gearSize;
attribute vec2 vertexPosition;

...

void main() {
    float instanceID = float(gl_InstanceIDAPPLE);
    instancePosition.x = mod(instanceID, 10) * gearSize;
    instancePosition.y = (floor(instanceID)/ 10) * gearSize;
    vec4 tempPos = vertexPosition.xy + instancePosition.xy;

    // Transform tempPos to clip space and output to gl_Position

    ...

    gl_Position = tempPos;
}
```

Shader Instance ID

Vertex shader


```
uniform float gearSize;
attribute vec2 vertexPosition;

...

void main() {
    float instanceID = float(gl_InstanceIDAPPLE);
    instancePosition.x = mod(instanceID, 10) * gearSize;
    instancePosition.y = (floor(instanceID)/ 10) * gearSize;
    vec4 tempPos = vertexPosition.xy + instancePosition.xy;

    // Transform tempPos to clip space and output to gl_Position

    ...

    gl_Position = tempPos;
}
```

Shader Instance ID

Vertex shader


```
uniform float gearSize;
attribute vec2 vertexPosition;

...

void main() {
    float instanceID = float(gl_InstanceIDAPPLE);
    instancePosition.x = mod(instanceID, 10) * gearSize;
    instancePosition.y = (floor(instanceID)/ 10) * gearSize;
    vec4 tempPos = vertexPosition.xy + instancePosition.xy;

    // Transform tempPos to clip space and output to gl_Position

    ...

    gl_Position = tempPos;
}
```

Vertex Texture Sampling

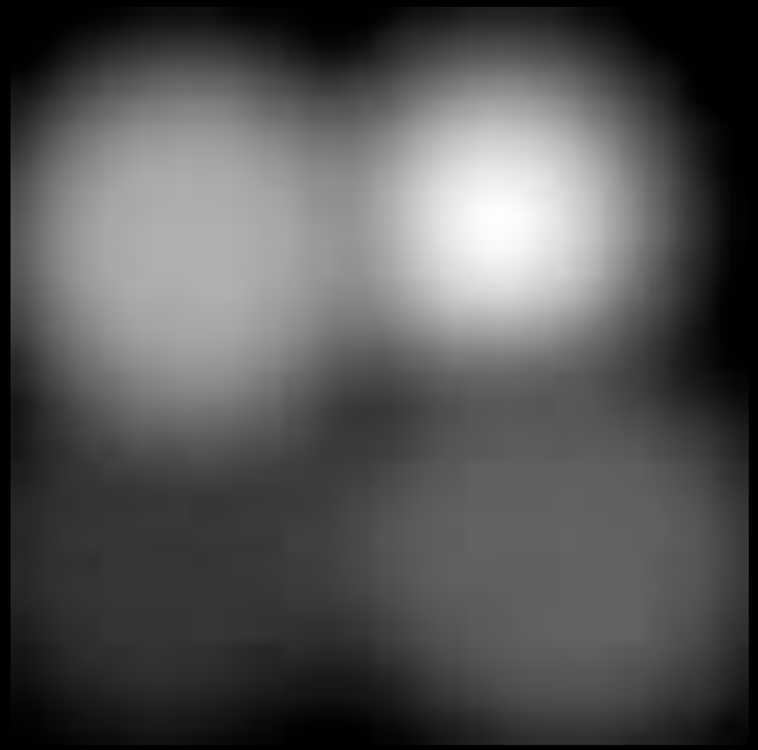
A texture in the vertex stage?

Vertex Texture Sampling

- Texture access in the vertex shader
- Many uses
  - Displacement mapping
  - Alternative to uniforms
- Generic data store with random access

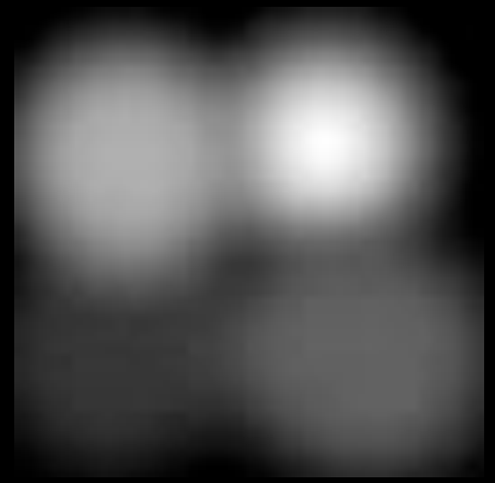
A Simple Vertex Texture Sampling Example

Height mapping



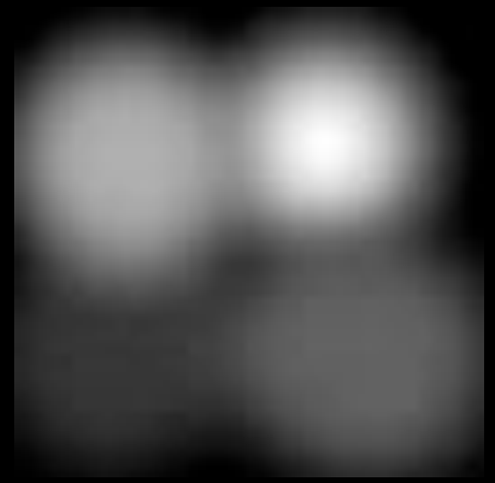
A Simple Vertex Texture Sampling Example

```
attribute vec2 xzPos;
uniform sampler2D heightMap;

...

main()
{
    vec4 tempPos = texture2D(heightMap, xzPos);
    tempPos.xz = xzPos;

    // Transform tempPos to from model space to
    // clip space with modelviewProjection matrix

    ...

    // output to gl_Position
    gl_Position = tempPos;
}
```



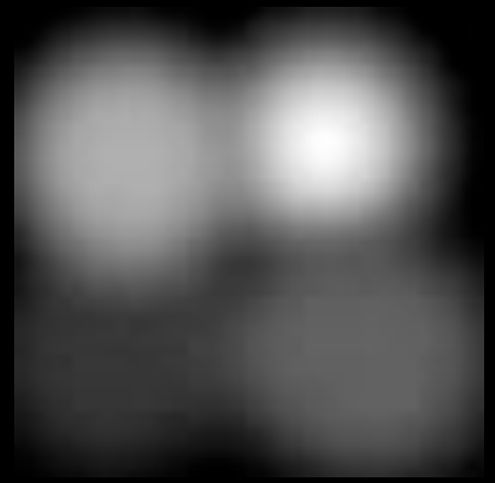
A Simple Vertex Texture Sampling Example

```
attribute vec2 xzPos;
uniform sampler2D heightMap;

...

main()
{
    vec4 tempPos = texture2D(heightMap, xzPos);
    tempPos.xz = xzPos;

    // Transform tempPos to from model space to
    // clip space with modelviewProjection matrix

    ...

    // output to gl_Position
    gl_Position = tempPos;
}
```



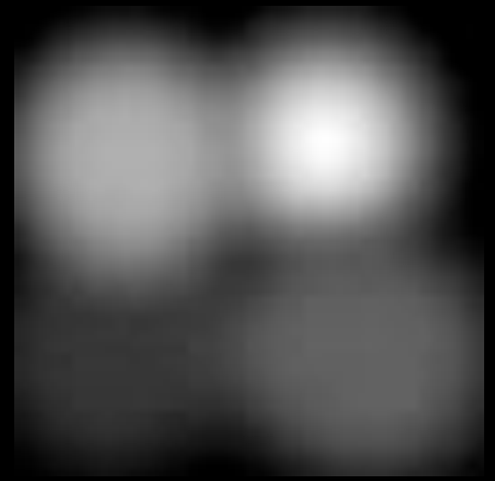
A Simple Vertex Texture Sampling Example

```
attribute vec2 xzPos;
uniform sampler2D heightMap;

...

main()
{
    vec4 tempPos = texture2D(heightMap, xzPos);
    tempPos.xz = xzPos;

    // Transform tempPos to from model space to
    // clip space with modelviewProjection matrix

    ...

    // output to gl_Position
    gl_Position = tempPos;
}
```



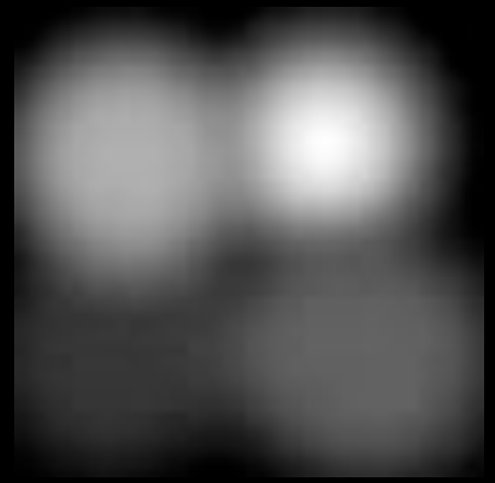
A Simple Vertex Texture Sampling Example

```
attribute vec2 xzPos;
uniform sampler2D heightMap;

...

main()
{
    vec4 tempPos = texture2D(heightMap, xzPos);
    tempPos.xz = xzPos;

    // Transform tempPos to from model space to
    // clip space with modelviewProjection matrix

    ...

    // output to gl_Position
    gl_Position = tempPos;
}
```



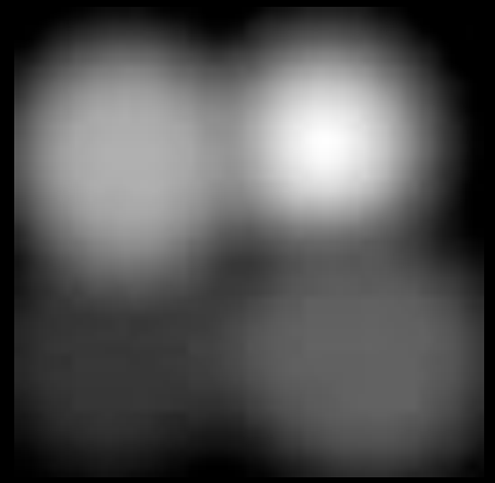
A Simple Vertex Texture Sampling Example

```
attribute vec2 xzPos;
uniform sampler2D heightMap;

...

main()
{
    vec4 tempPos = texture2D(heightMap, xzPos);
    tempPos.xz = xzPos;

    // Transform tempPos to from model space to
    // clip space with modelviewProjection matrix

    ...

    // output to gl_Position
    gl_Position = tempPos;
}
```



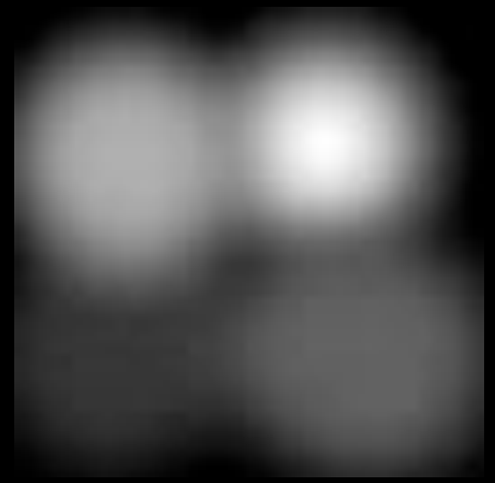
A Simple Vertex Texture Sampling Example

```
attribute vec2 xzPos;
uniform sampler2D heightMap;

...

main()
{
    vec4 tempPos = texture2D(heightMap, xzPos);
    tempPos.xz = xzPos;

    // Transform tempPos to from model space to
    // clip space with modelviewProjection matrix

    ...

    // output to gl_Position
    gl_Position = tempPos;
}
```



A Simple Vertex Texture Sampling Example

```
attribute vec2 xzPos;
uniform sampler2D heightMap;

...

main()
{
    vec4 tempPos = texture2D(heightMap, xzPos);
    tempPos.xz = xzPos;

    // Transform tempPos to from model space to
    // clip space with modelviewProjection matrix

    ...

    // output to gl_Position
    gl_Position = tempPos;
}
```



Vertex Texture Sampling for Generic Data

- Can store any kind of data in a texture for shader access
  - Large store of random access memory
  - Data normally passed via `glUniform`, passed via texture

Texture Sampling vs. Uniforms

- Larger storage than a uniform array
- Potentially less API calls to set data
  - Bind large number of uniforms as group
- A variety of types
  - `GL_UNSIGNED_BYTE`, `GL_HALF_FLOAT`, and `GL_FLOAT`
- Use with filtering and wrapping
  - Averages sequential values
- Render to texture
  - GPU can produce data

Demo

Instanced Asteroids

Instanced Asteroids

Instancing with shader instance ID

- Calculates transformation matrix in vertex shader
  - `gl_InstanceIDAPPLE` modded by a constant gives a spin value
    - cos and sin used to generate rotation matrix
  - `gl_InstanceIDAPPLE` also used to calculate position
    - Translation applied to rotation matrix
- Matrix calculations are done per vertex

Instanced Asteroids

Instancing with instanced arrays

- Two vertex arrays store positions and rotations
  - Calculated once at app startup
- Attribute divisor passes parameters for each asteroid
  - Not passed per vertex

Instanced Asteroids

Instance ID with texture sampling vs. instanced arrays

- Advantages of instance ID
  - Little memory or bandwidth required
  - Use GPU for computation

Instanced Asteroids

Instance ID with texture sampling vs. instanced arrays

- Advantages of instance ID
  - Little memory or bandwidth required
  - Use GPU for computation
- Advantages of instanced arrays
  - Generally faster than computing on GPU
  - Lots of flexibility in types

Instanced Asteroids

Instance ID with texture sampling vs. instanced arrays

- Advantages of instance ID
  - Little memory or bandwidth required
  - Use GPU for computation
- Advantages of instanced arrays
  - Generally faster than computing on GPU
  - Lots of flexibility in types
- Advantages of using instance ID with vertex texture sampling
  - Random access
  - Often logically simplest to store data
    - Look up tables, bone matrices

Instancing and Vertex Texture Sampling

Summary

- Instancing
  - Many models, single draw
  - Different parameters each instance
  - Performance gains
- Vertex texture sampling
  - Texture data access in vertex shader
  - Use with instance ID for per-instance parameters
- Available on all iOS 7 devices

sRGB

The other colorspace

sRGB Texture Formats

Gamma encoded texture formats

NEW

- sRGB is an alternate colorspace
- More perceptually correct
  - Matches gamma curve of displays
- Perceptually linear color mixing
  - Mixing with RGB weighs brighter colors more than darker ones

sRGB

Linear

sRGB Texture Formats

In the API

- New formats
  - `GL_SRGB_EXT           // External Format`
  - `GL_SRGB_ALPHA_EXT     // External Format`
  - `GL_SRGB8_ALPHA8_EXT   // Internal Format`
  - `GL_COMPRESSED_SRGB_PVRTC_..._APPLE // Compressed Formats`
- Noncompressed sRGB textures are renderable
  - Linear blending
  - Color calculations in shaders
- Check for `GL_EXT_sRGB` in the extension string
  - Not supported on iPhone 4

sRGB Texture Formats

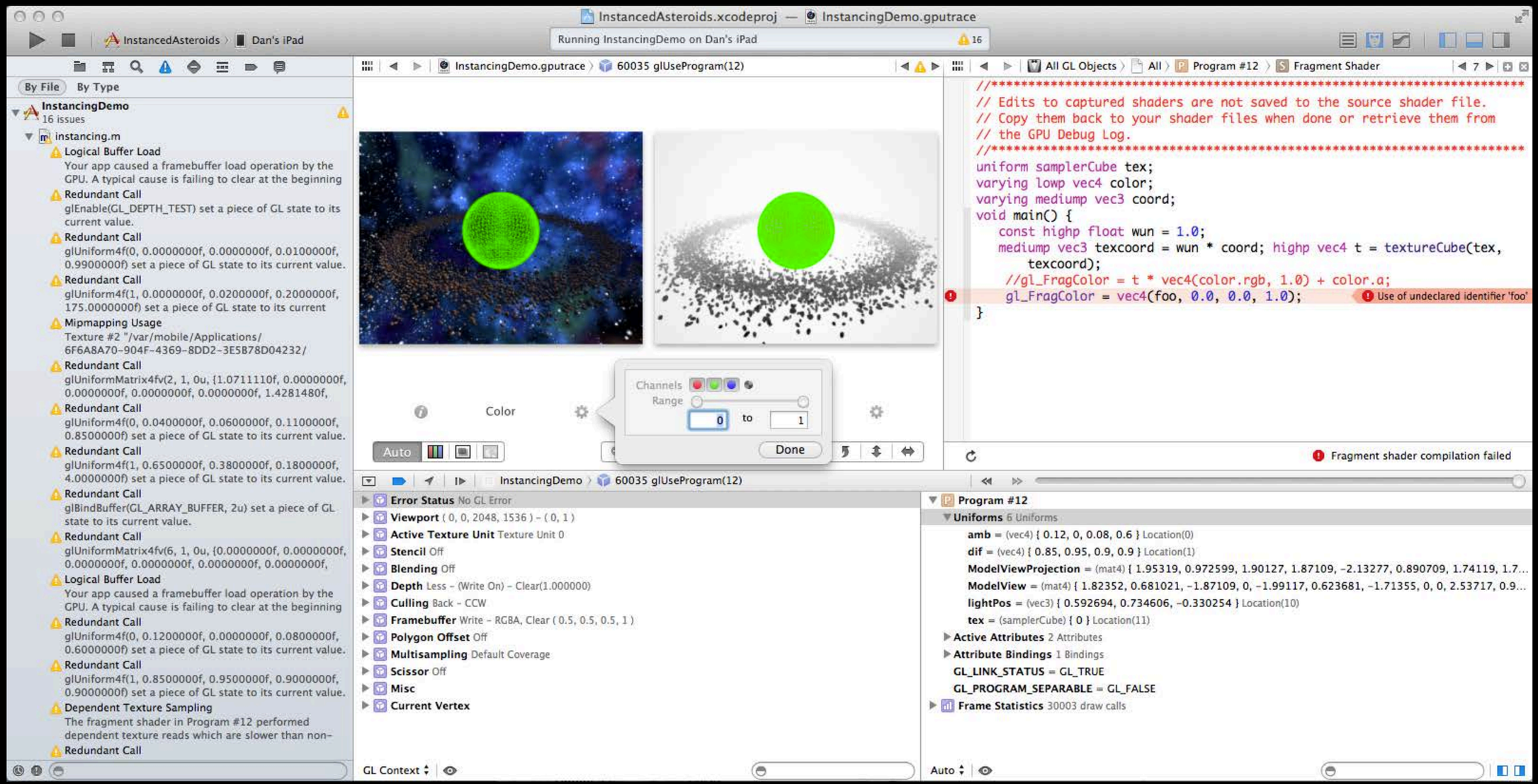
When they should be used

- The perceptually "correct" colorspace to use for mixing
- Author texture images with sRGB colorspace in mind
  - Otherwise may look different than intended
- Only use for color data

GPU Tools

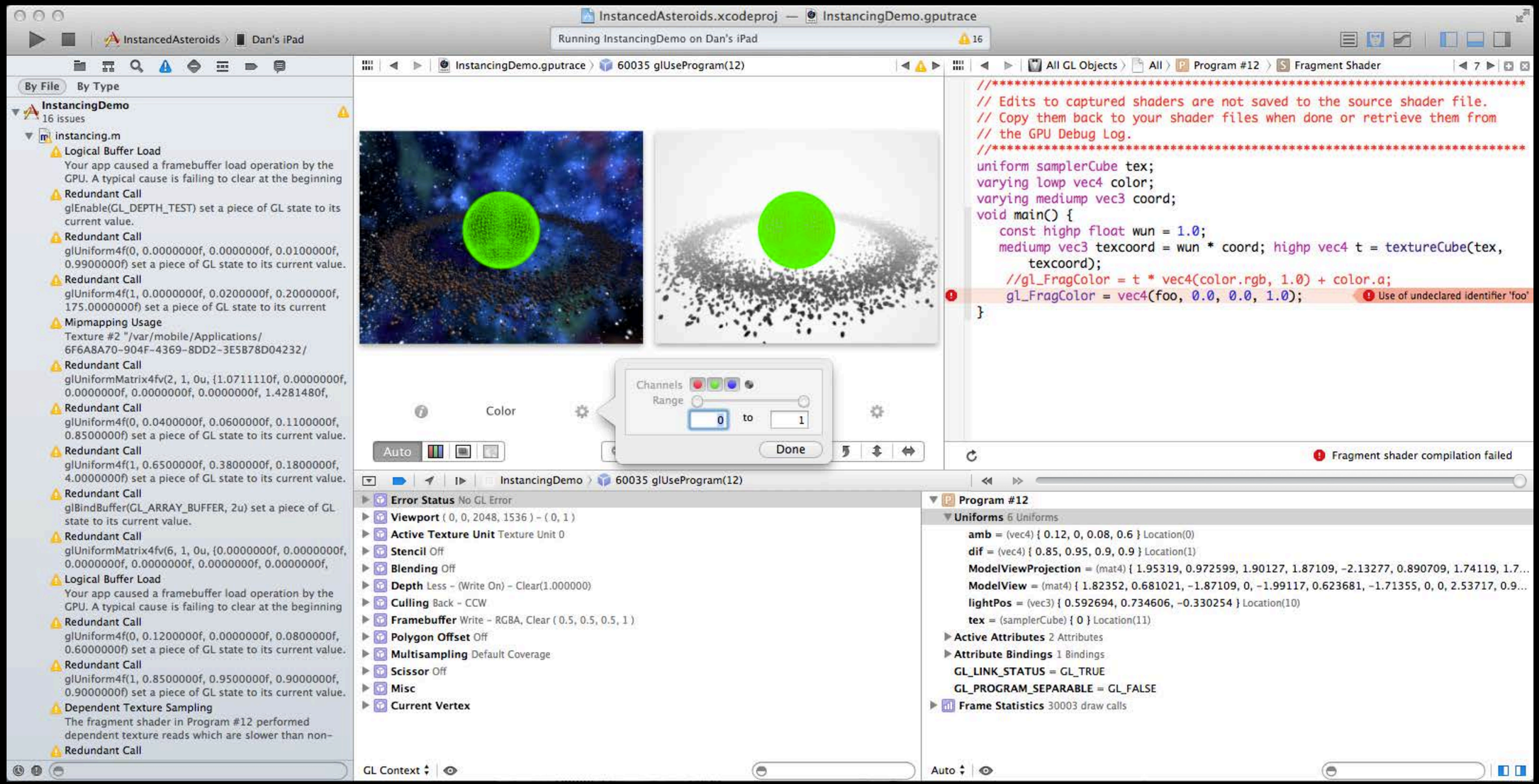
Building a solid foundation

OpenGL ES Frame Debugger



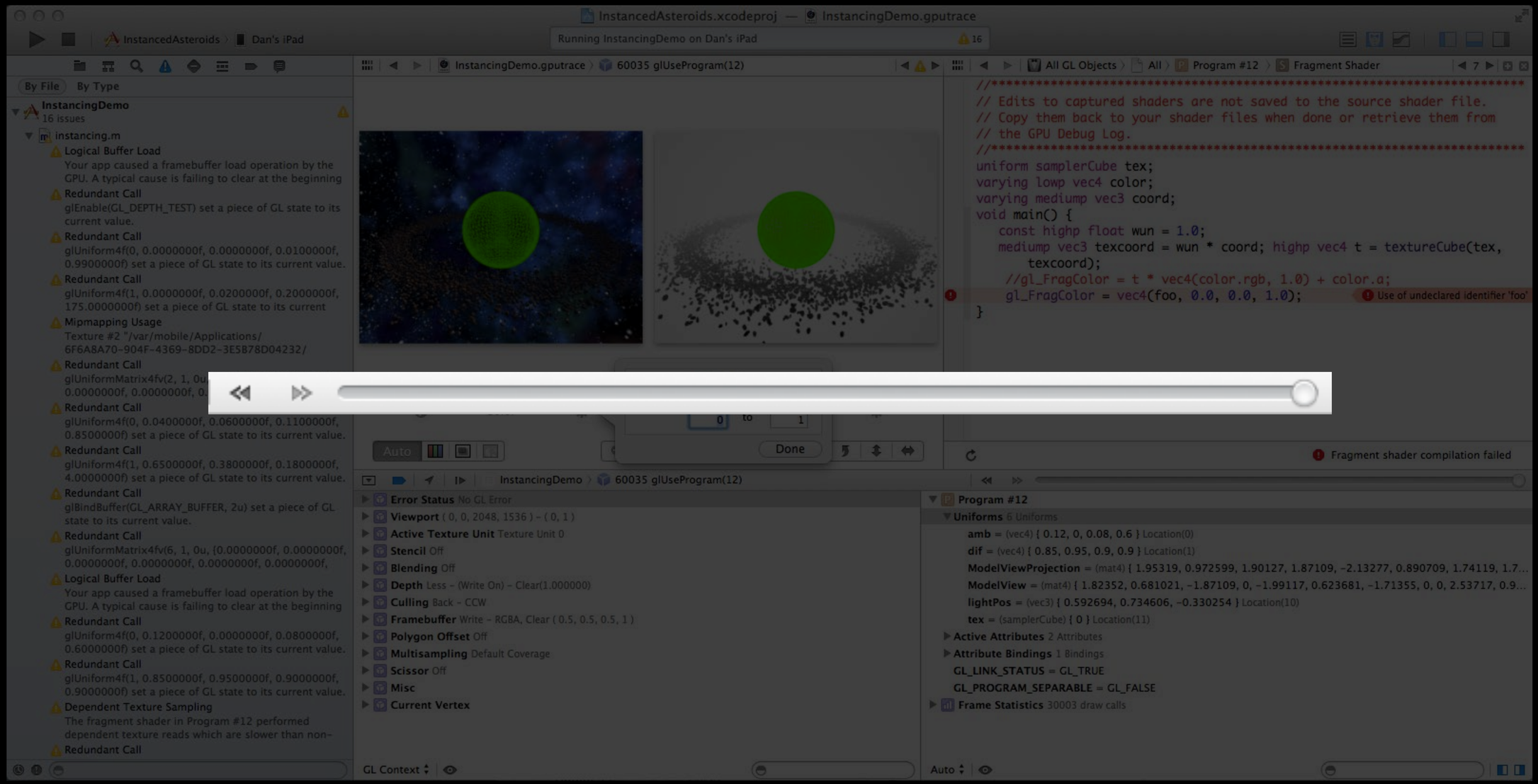
OpenGL ES Frame Debugger

Scrubber bar and Framebuffer view



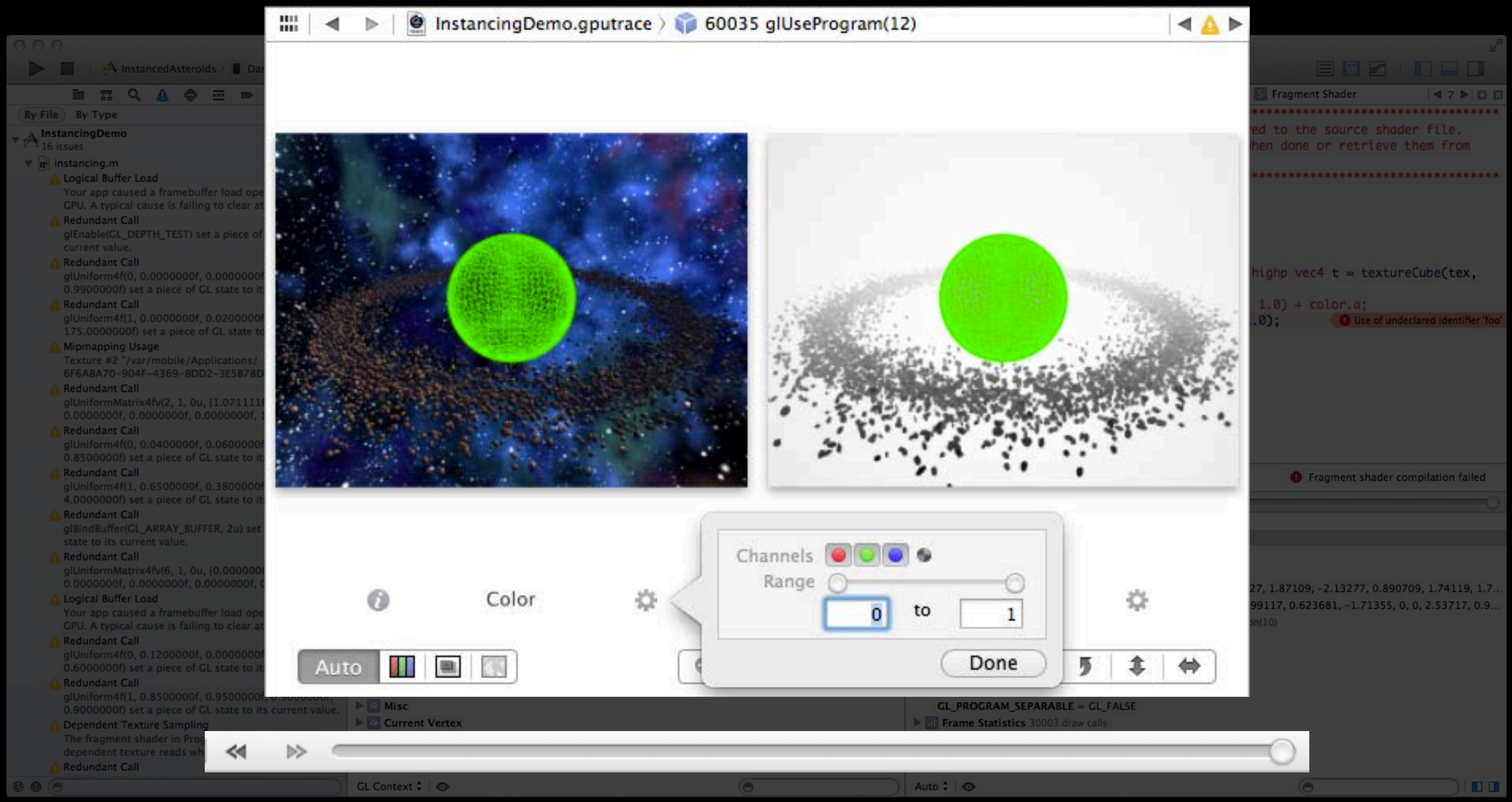
OpenGL ES Frame Debugger

Scrubber bar and Framebuffer view



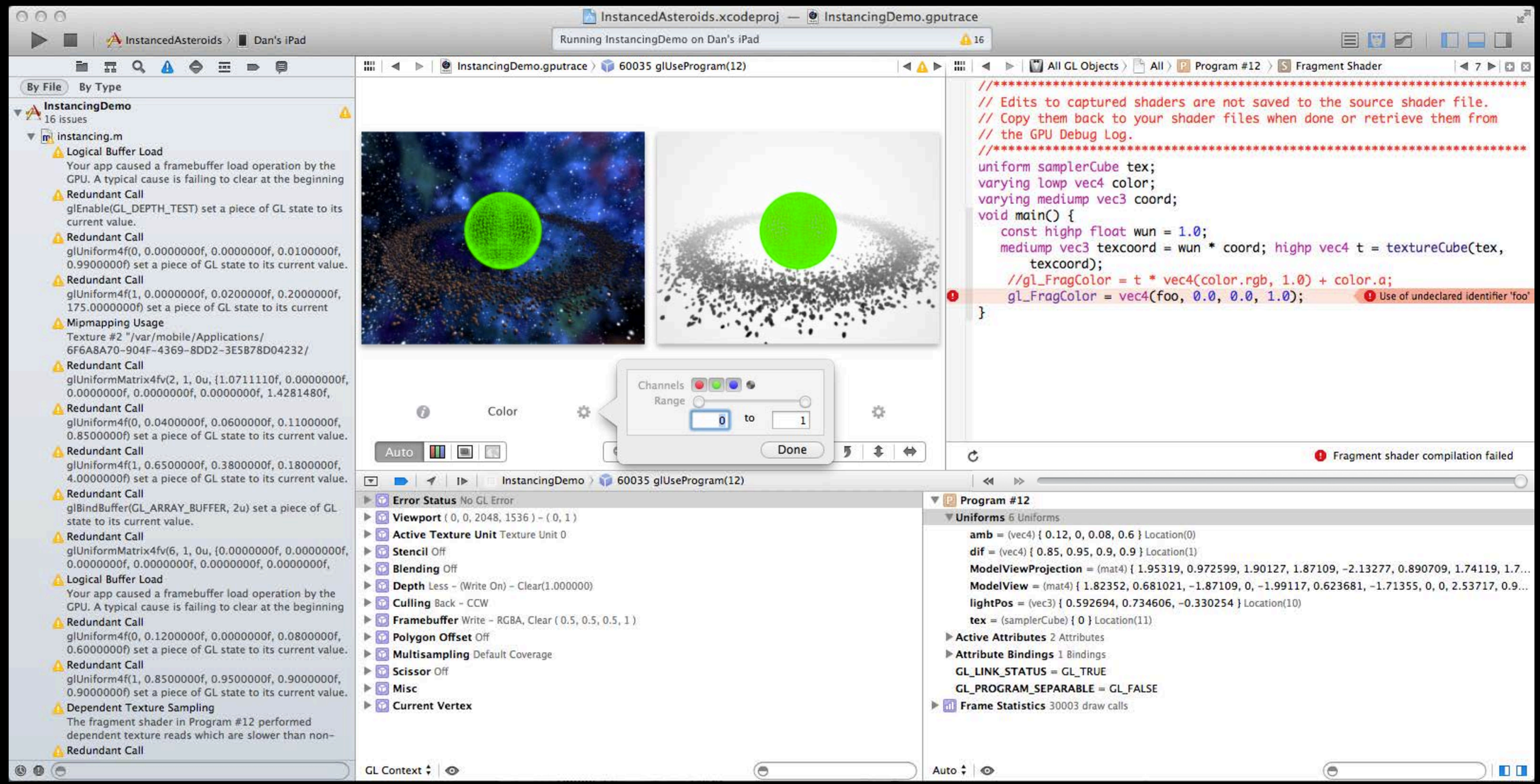
OpenGL ES Frame Debugger

Scrubber bar and Framebuffer view



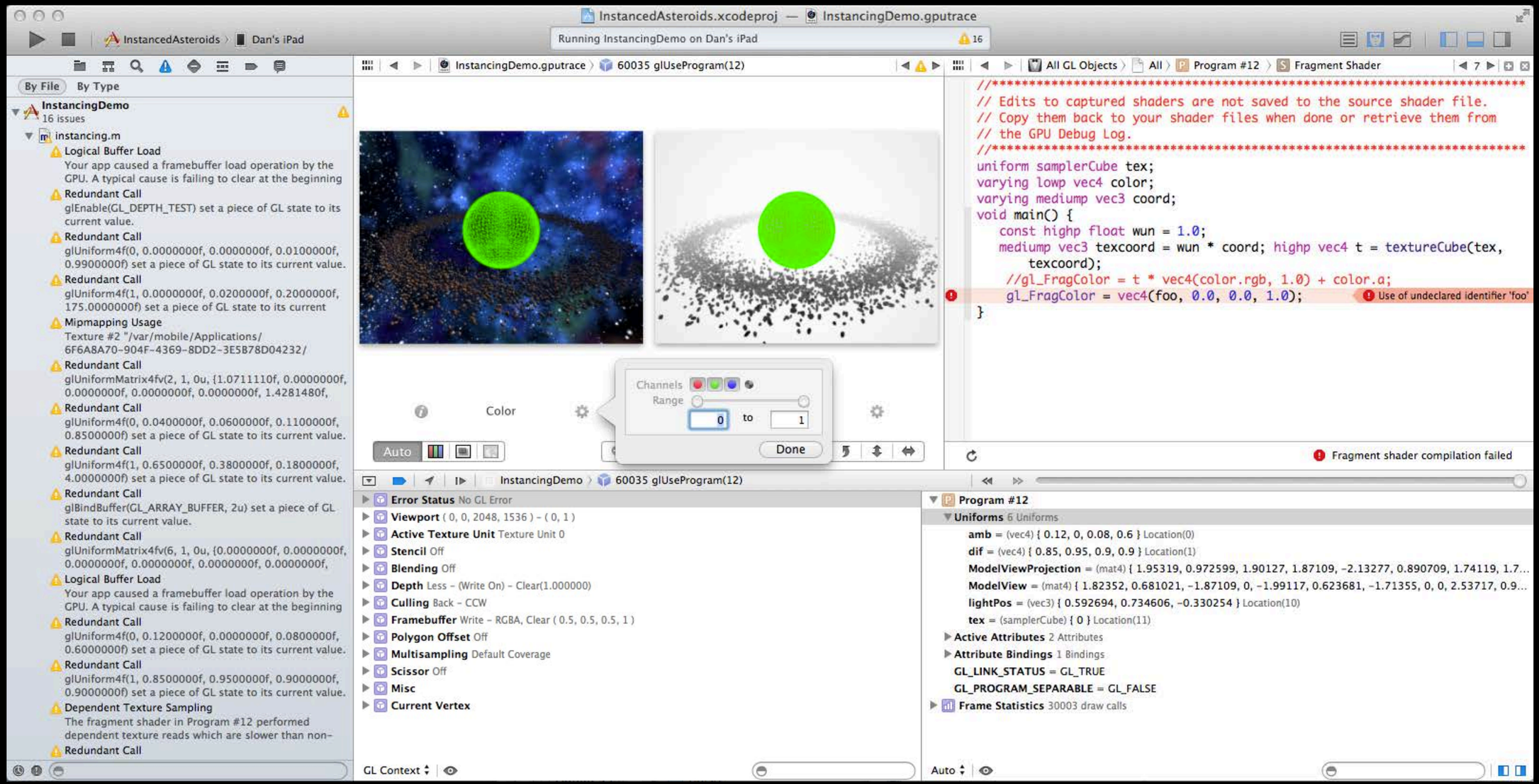
OpenGL ES Frame Debugger

Scrubber bar and Framebuffer view



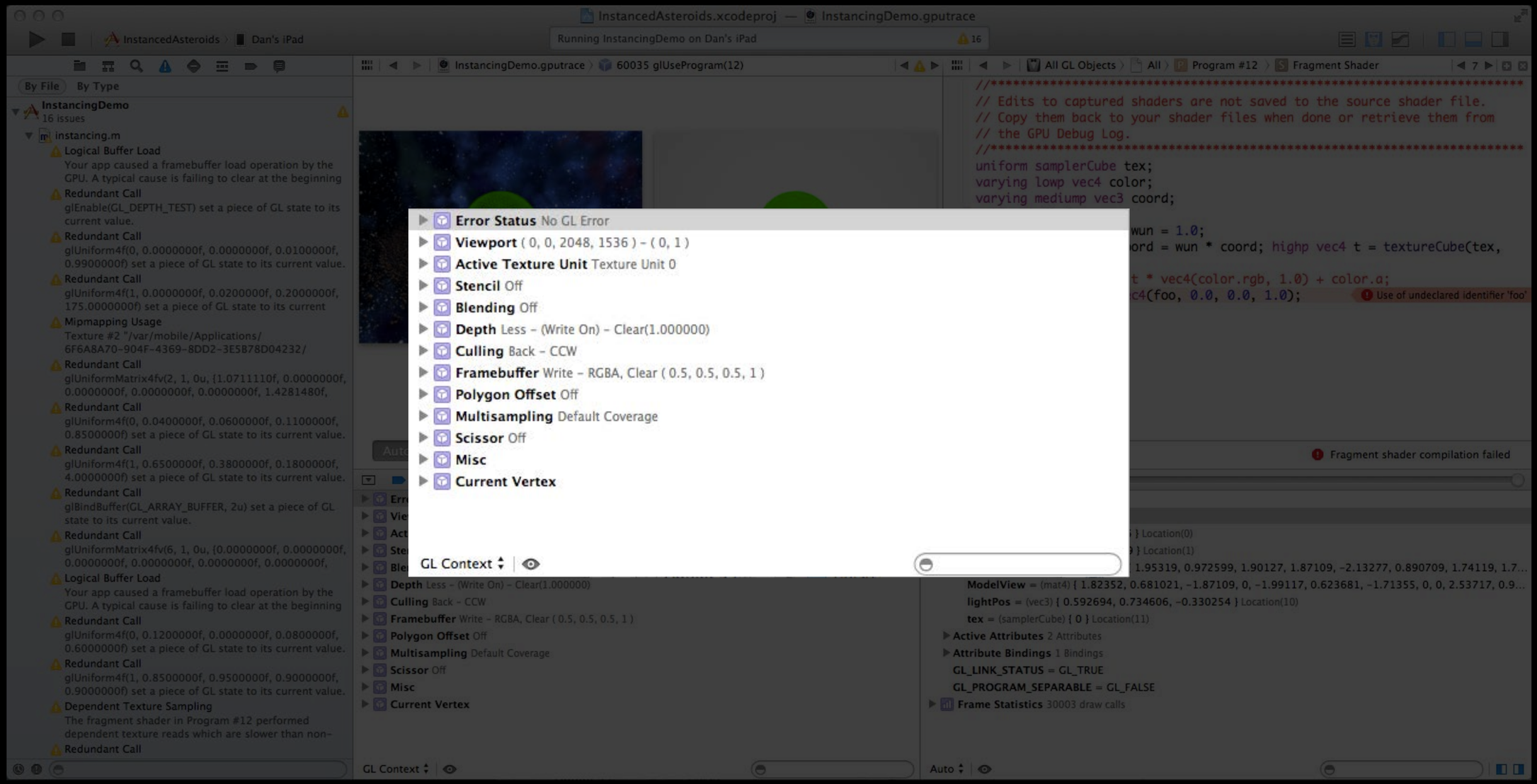
OpenGL ES Frame Debugger

GL context state and auto context view



OpenGL ES Frame Debugger

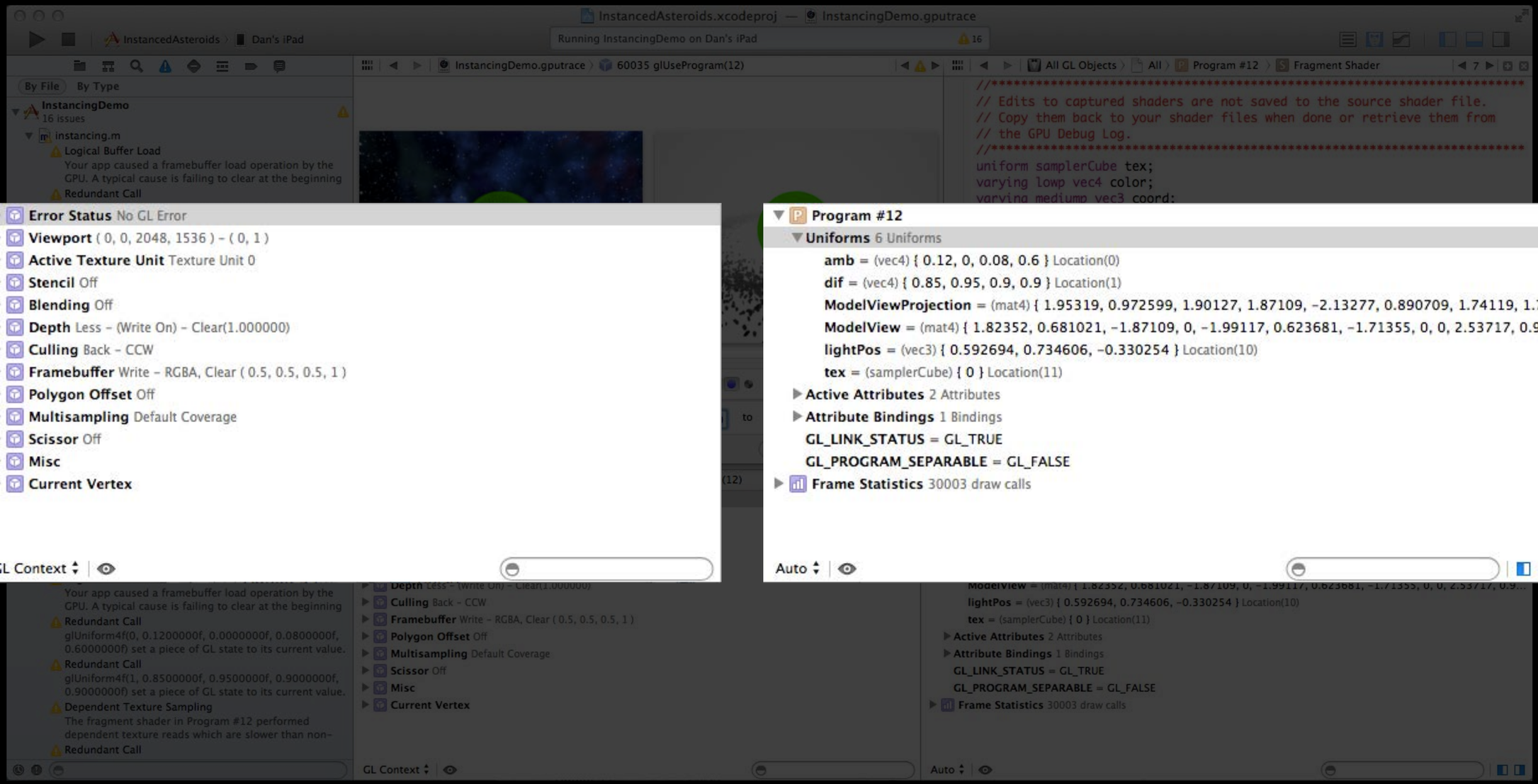
GL context state and auto context view



OpenGL ES Frame Debugger

GL context state and auto context view

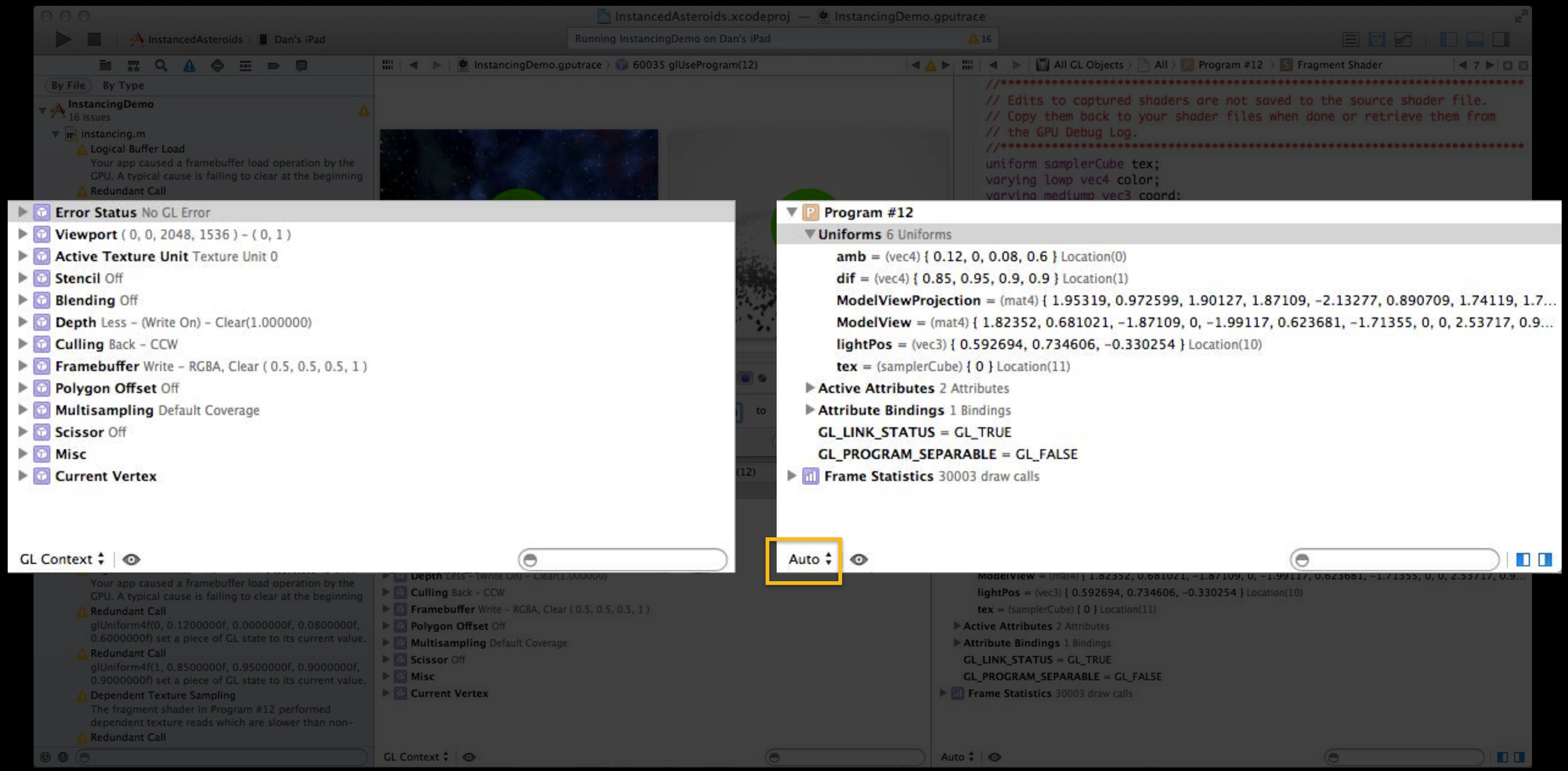
NEW



OpenGL ES Frame Debugger

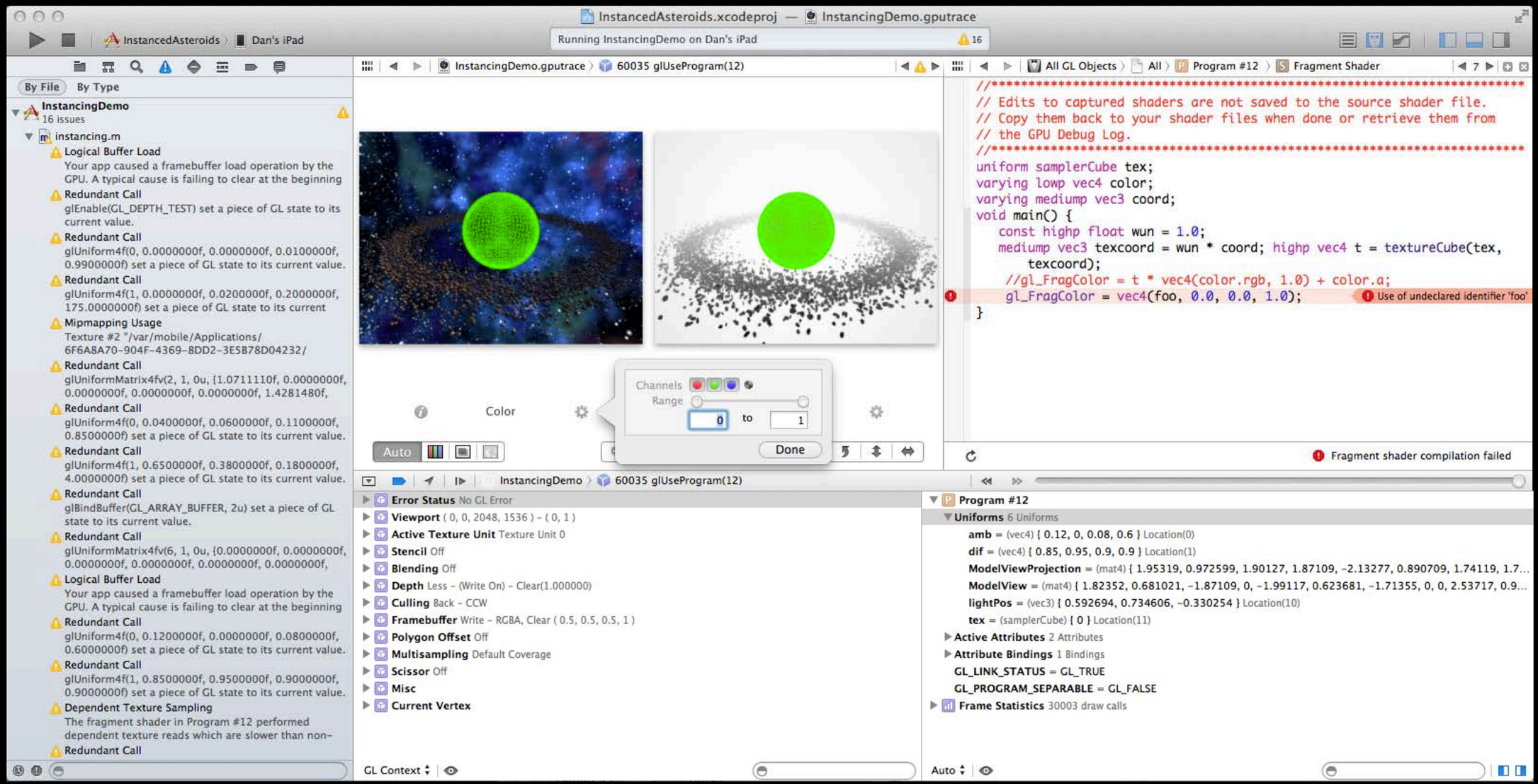
GL context state and auto context view

NEW



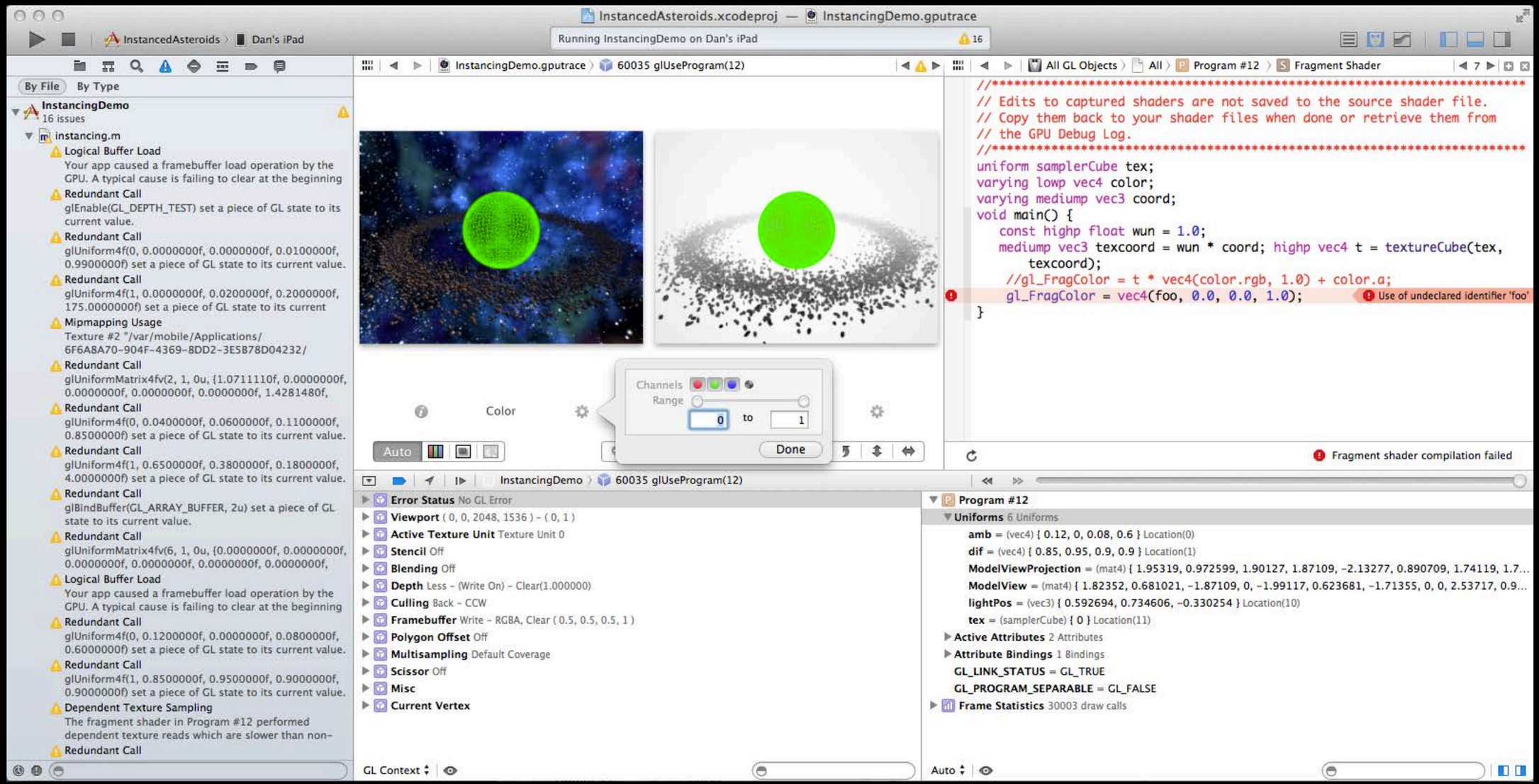
OpenGL ES Frame Debugger

GL context state and auto context view



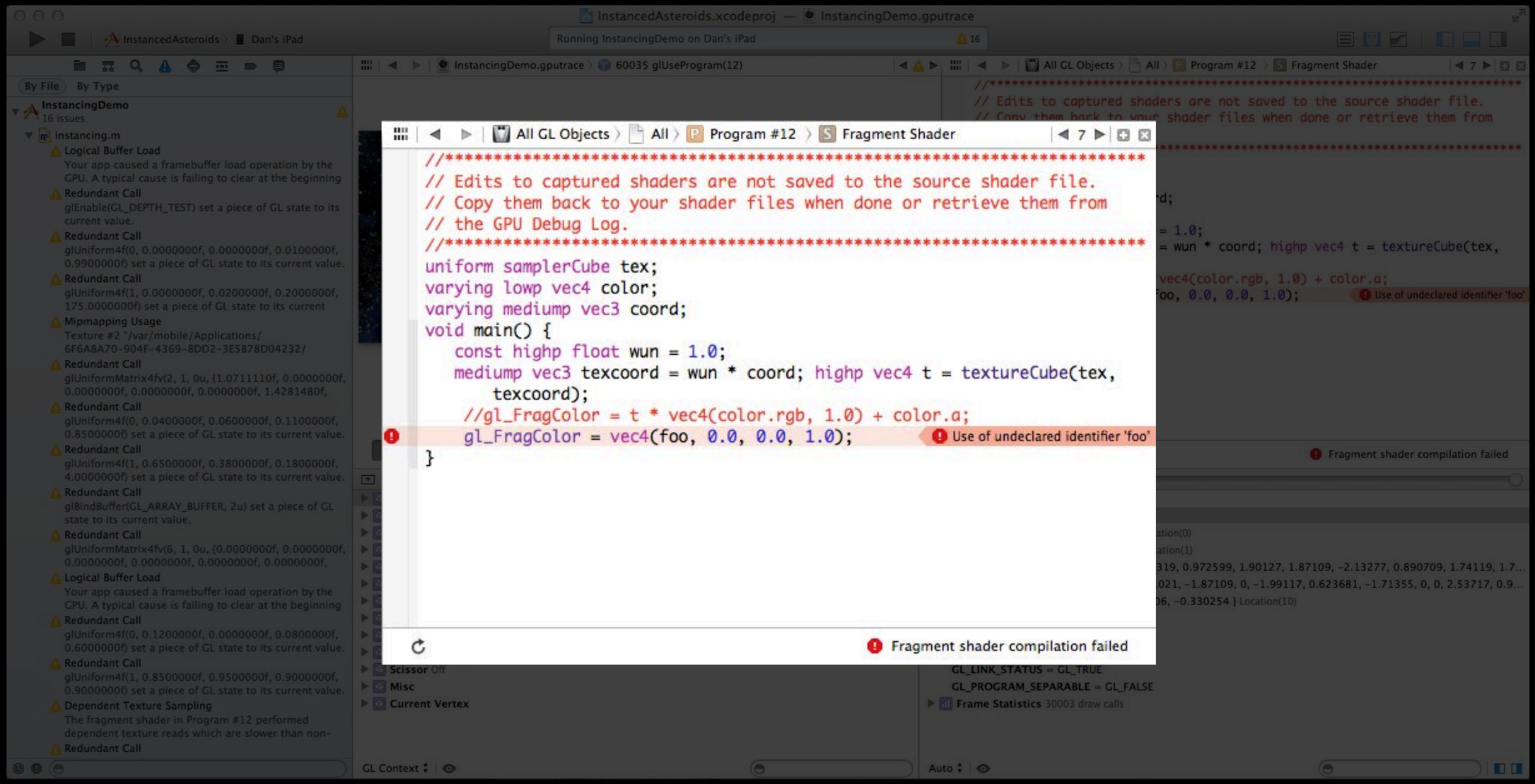
OpenGL ES Frame Debugger

Object view and Shader editor



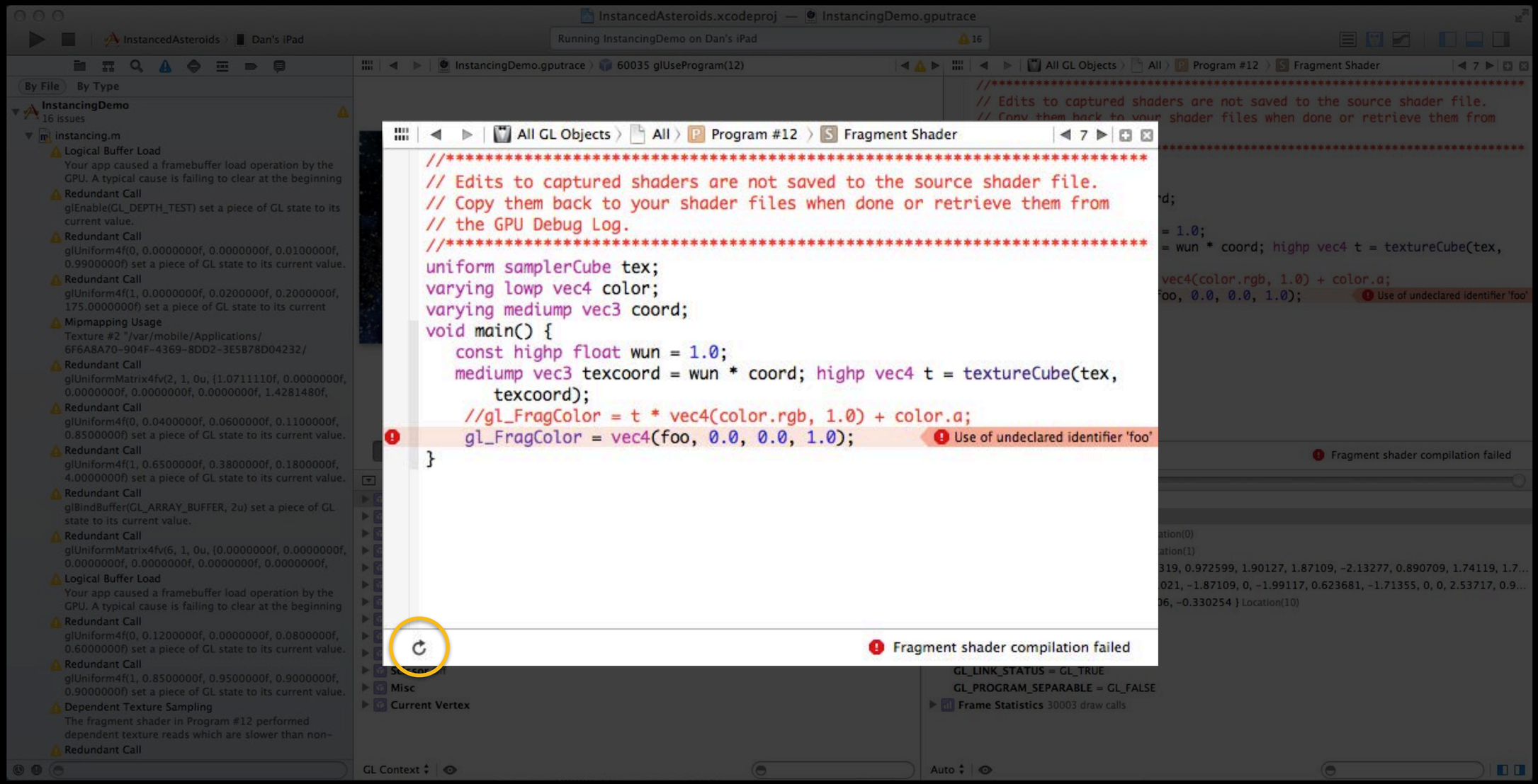
OpenGL ES Frame Debugger

Object view and Shader editor



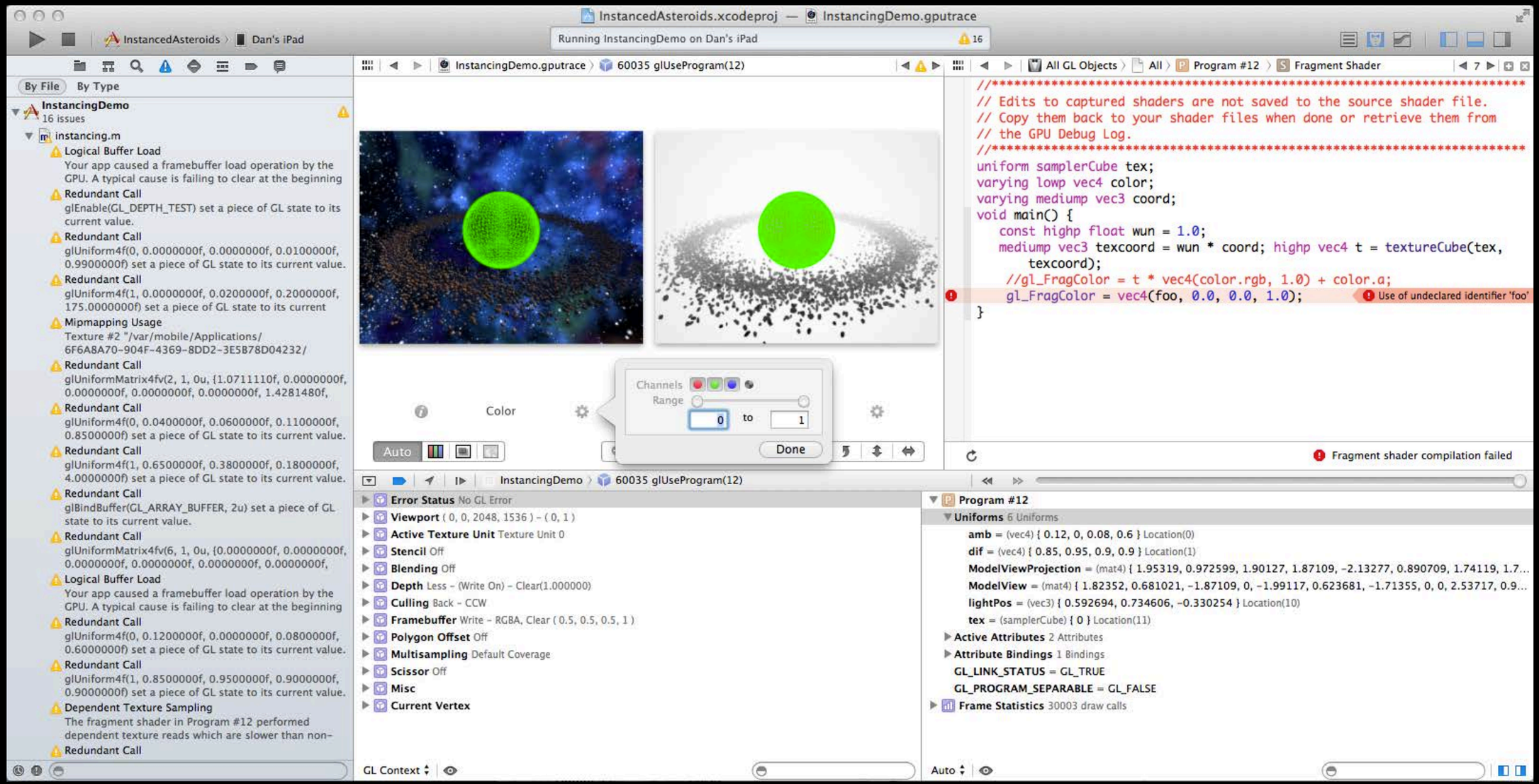
OpenGL ES Frame Debugger

Object view and Shader editor



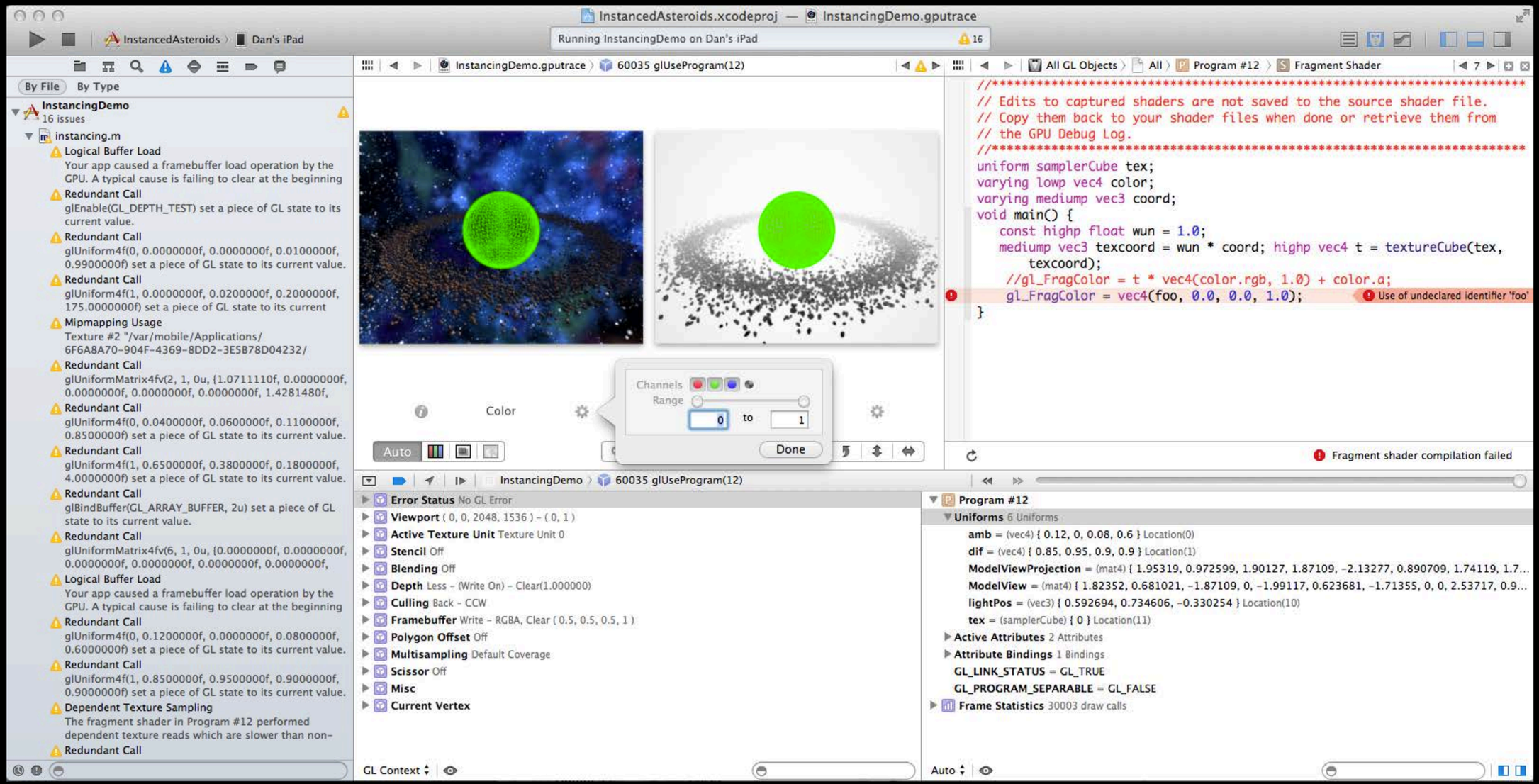
OpenGL ES Frame Debugger

Object view and Shader editor



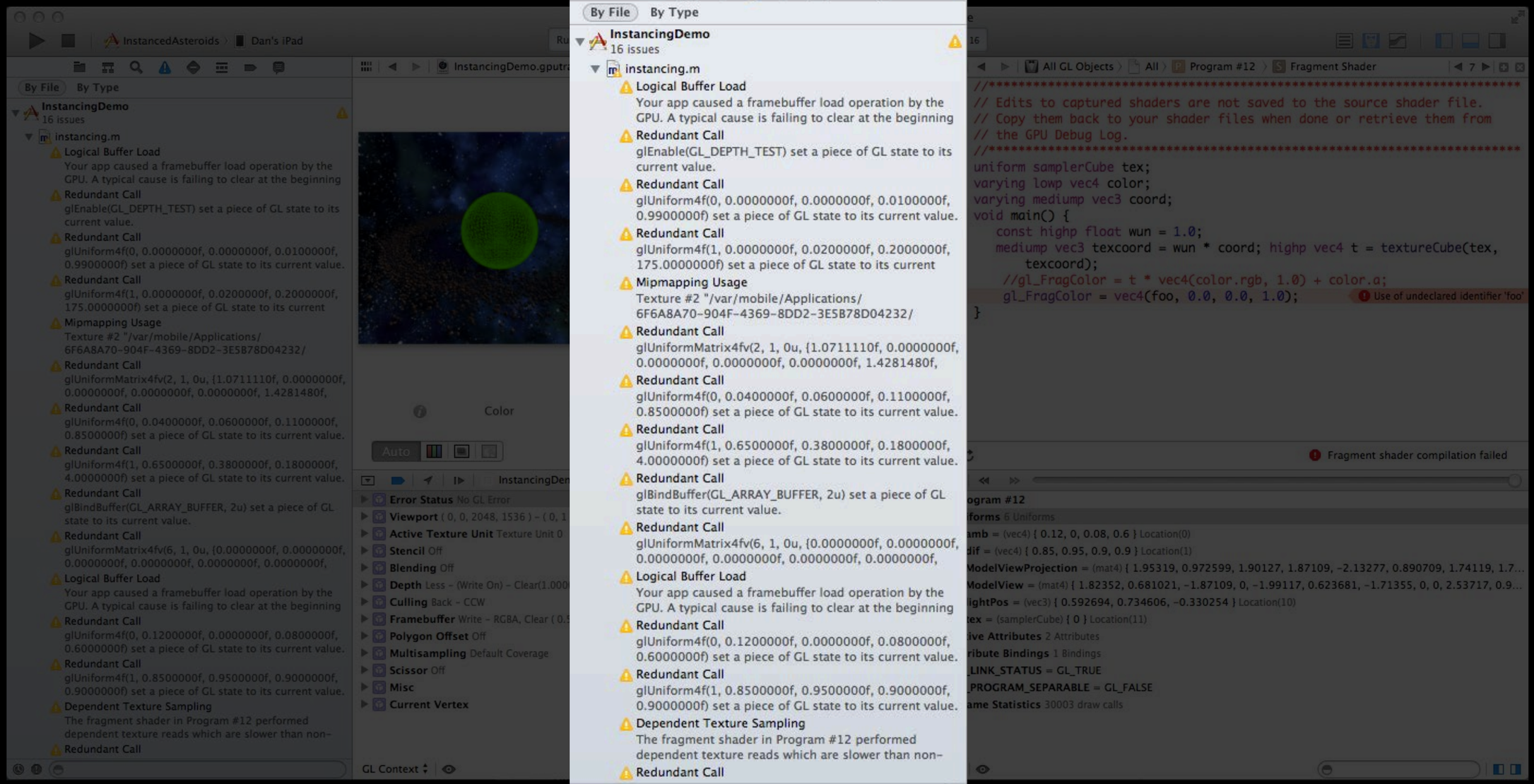
OpenGL ES Frame Debugger

GL issues navigator



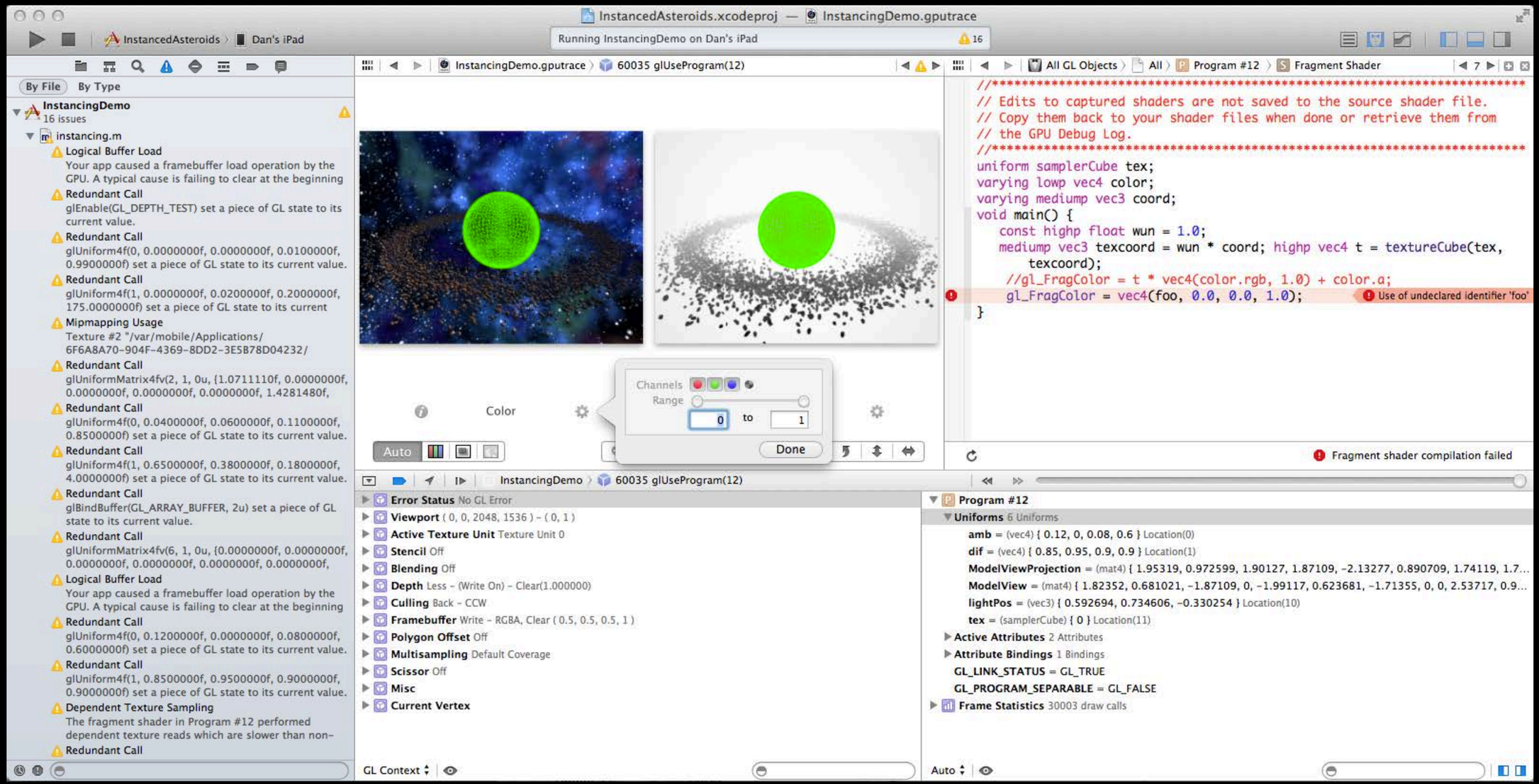
OpenGL ES Frame Debugger

GL issues navigator



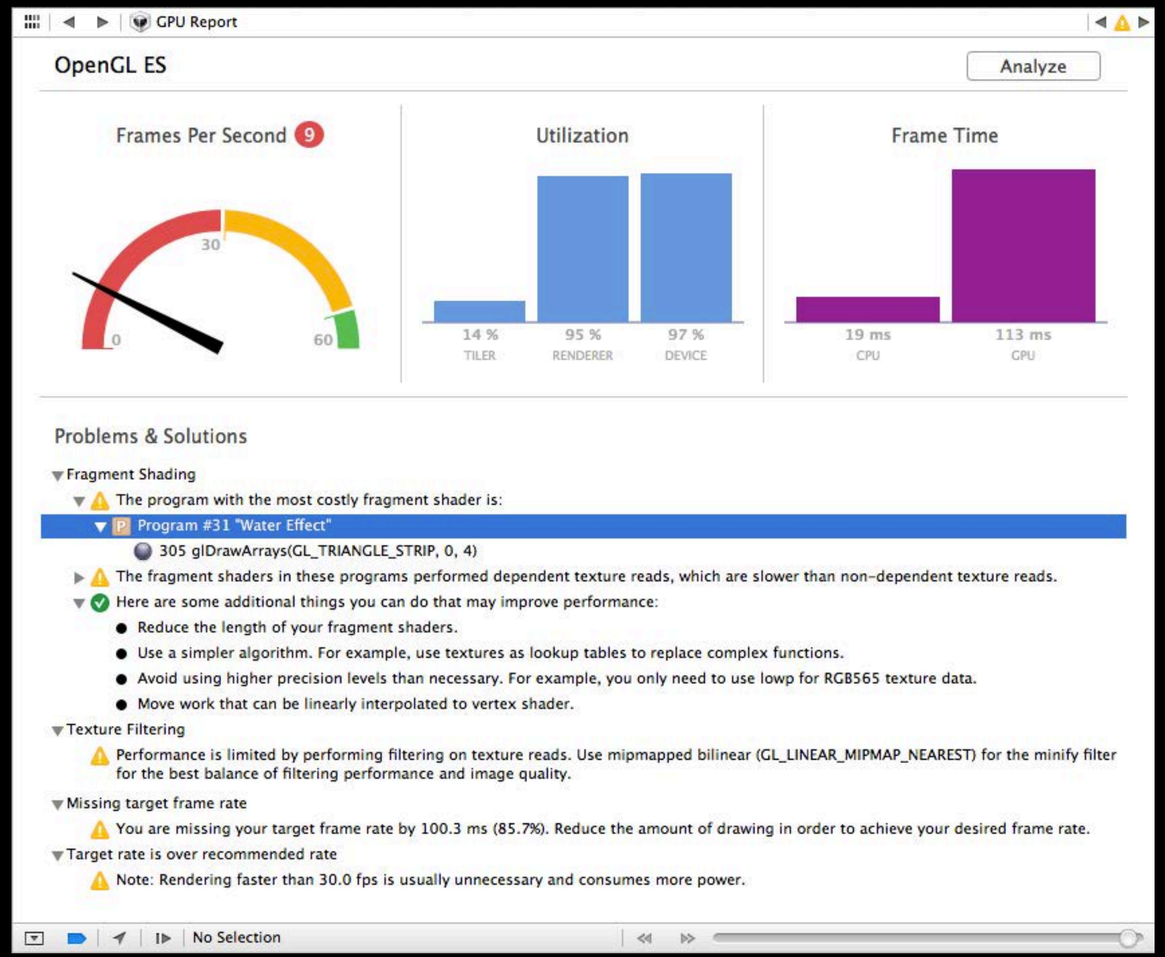
OpenGL ES Frame Debugger

GL issues navigator



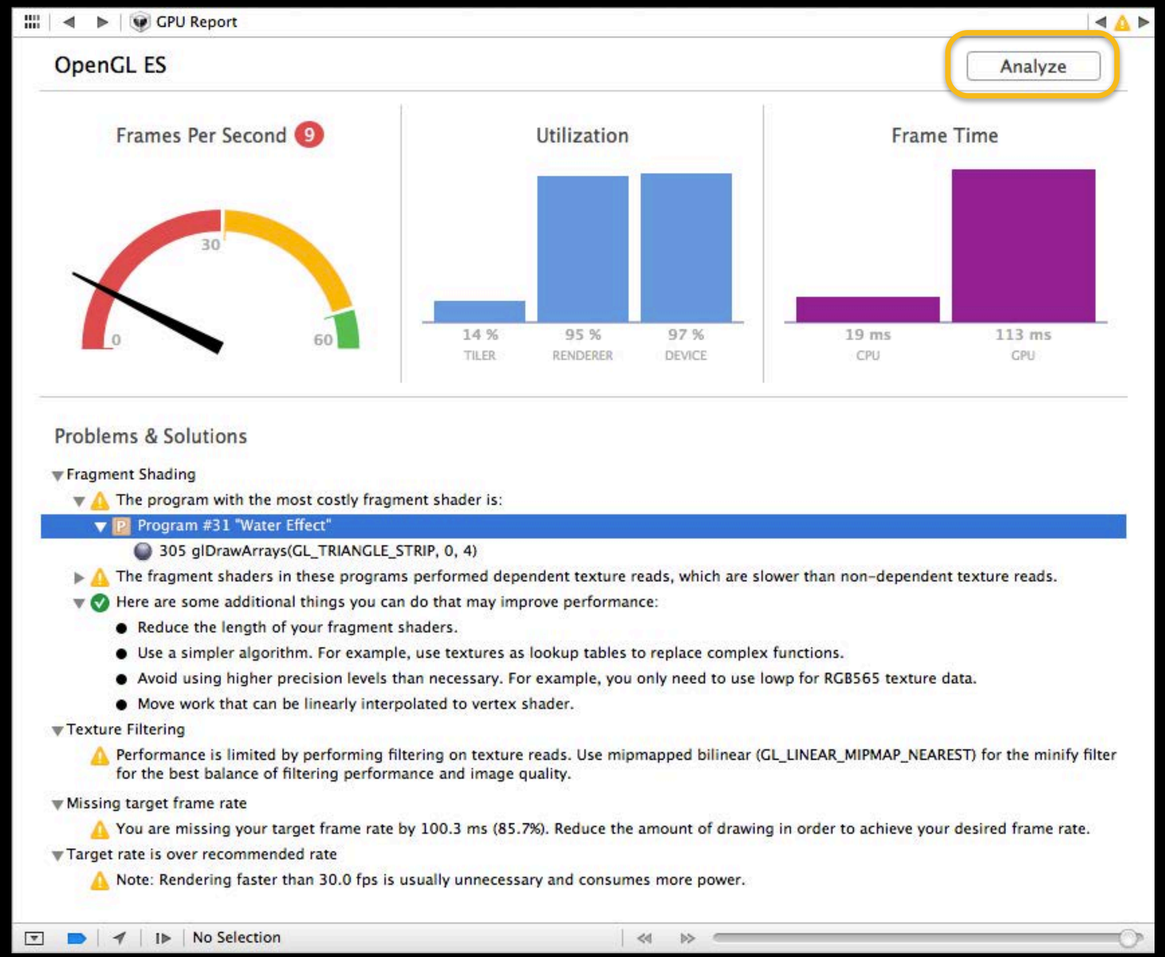
OpenGL ES Frame Debugger

OpenGL ES performance analysis



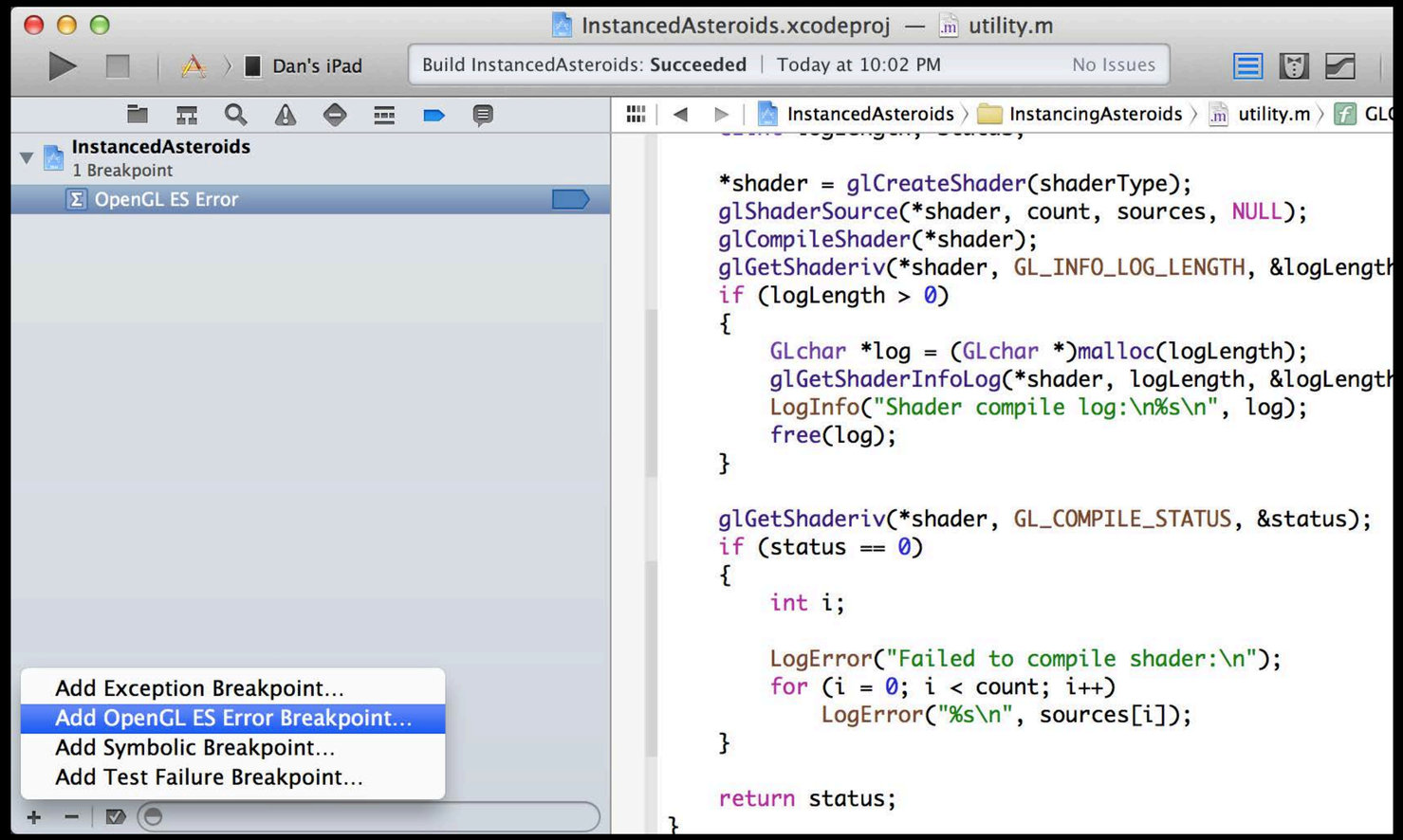
OpenGL ES Frame Debugger

OpenGL ES performance analysis

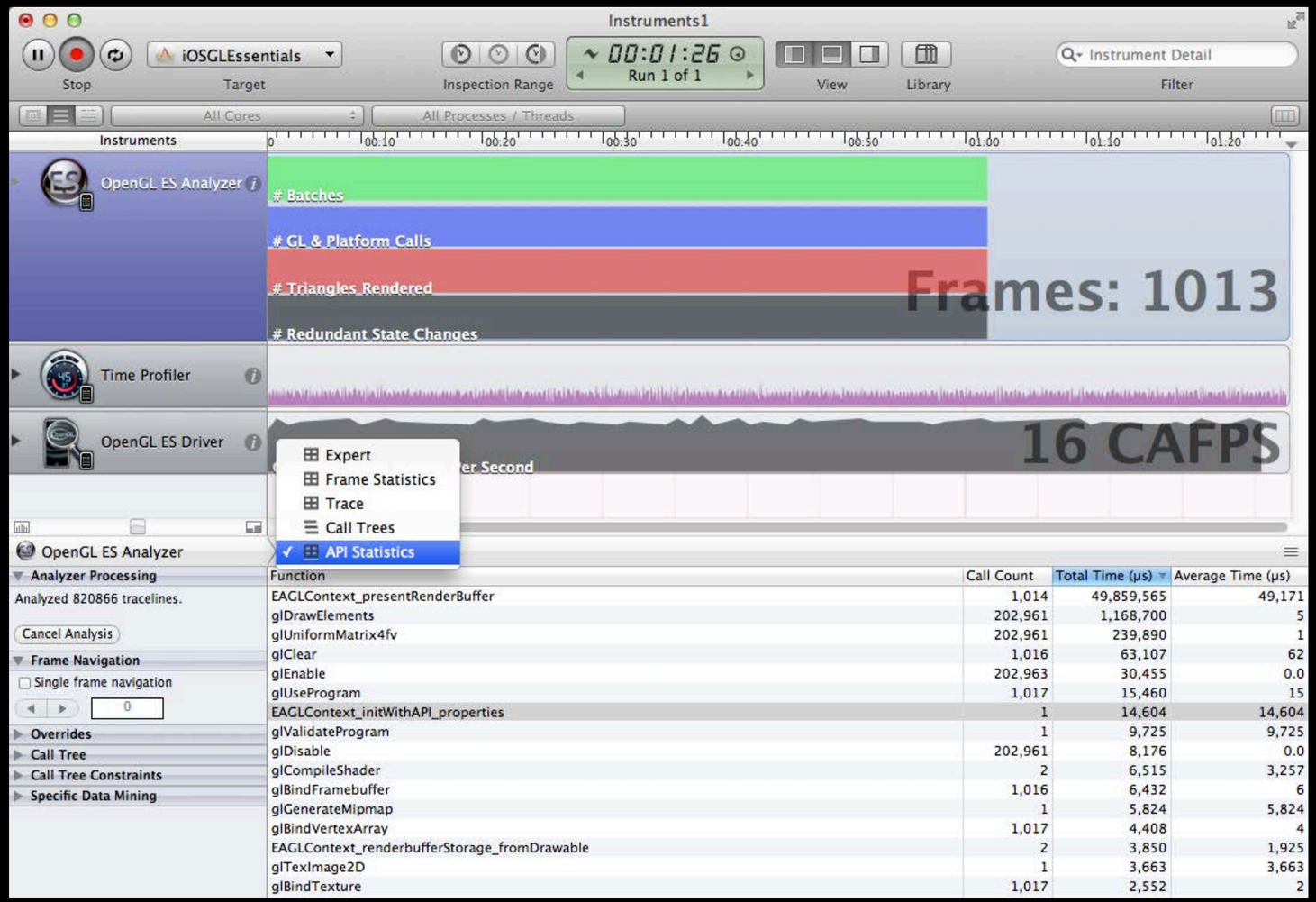


Xcode Breakpoints

Break on OpenGL ES error



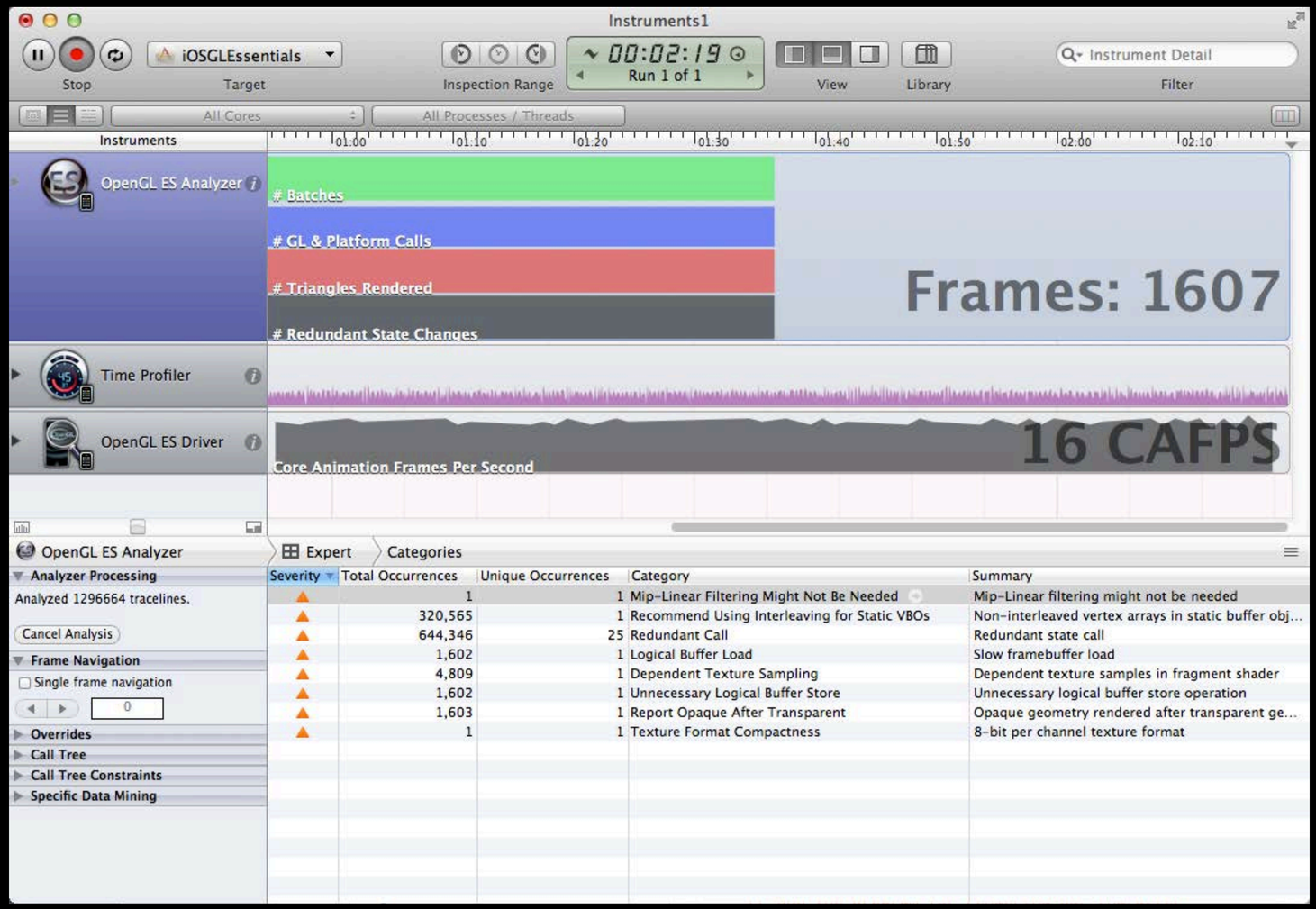
OpenGL ES Analyzer Instrument



Function	Call Count	Total Time (µs)	Average Time (µs)
EAGLContext_presentRenderBuffer	1,014	49,859,565	49,171
glDrawElements	202,961	1,168,700	5
glUniformMatrix4fv	202,961	239,890	1
glClear	1,016	63,107	62
glEnable	202,963	30,455	0.0
glUseProgram	1,017	15,460	15
EAGLContext_initWithAPI_properties	1	14,604	14,604
glValidateProgram	1	9,725	9,725
glDisable	202,961	8,176	0.0
glCompileShader	2	6,515	3,257
glBindFramebuffer	1,016	6,432	6
glGenerateMipmap	1	5,824	5,824
glBindVertexArray	1,017	4,408	4
EAGLContext_renderbufferStorage_fromDrawable	2	3,850	1,925
glTexImage2D	1	3,663	3,663
glBindTexture	1,017	2,552	2

OpenGL ES Analyzer Instrument

OpenGL ES expert



Maximizing GPU Performance

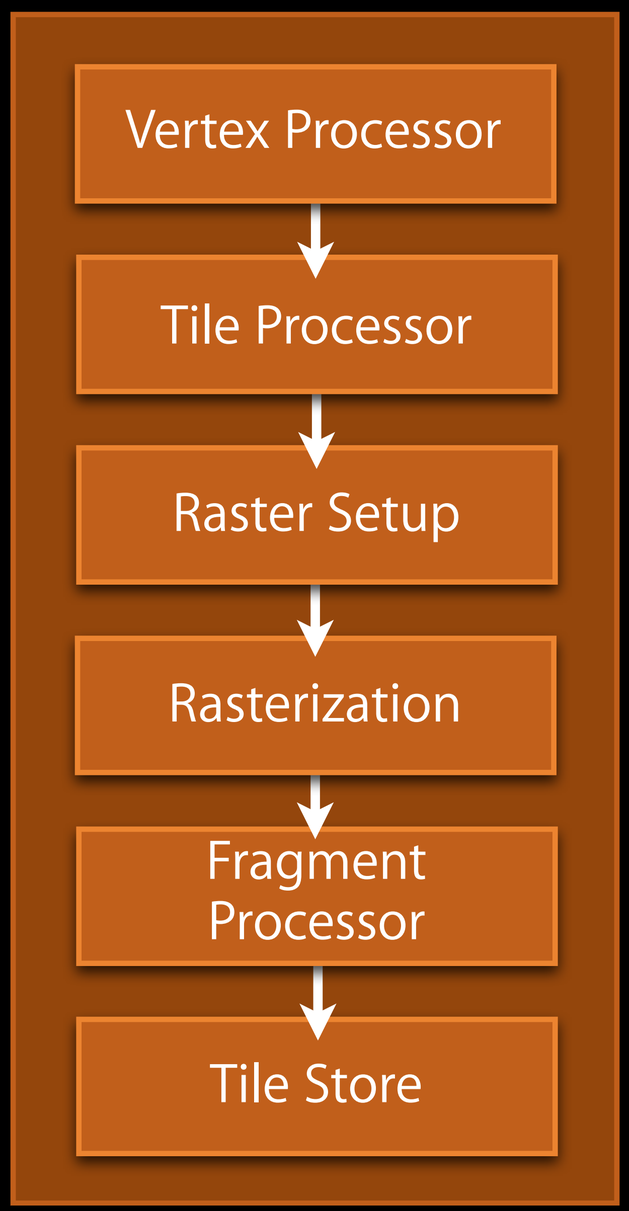
Understanding the GPU architecture

Tile-Based Deferred Renderer

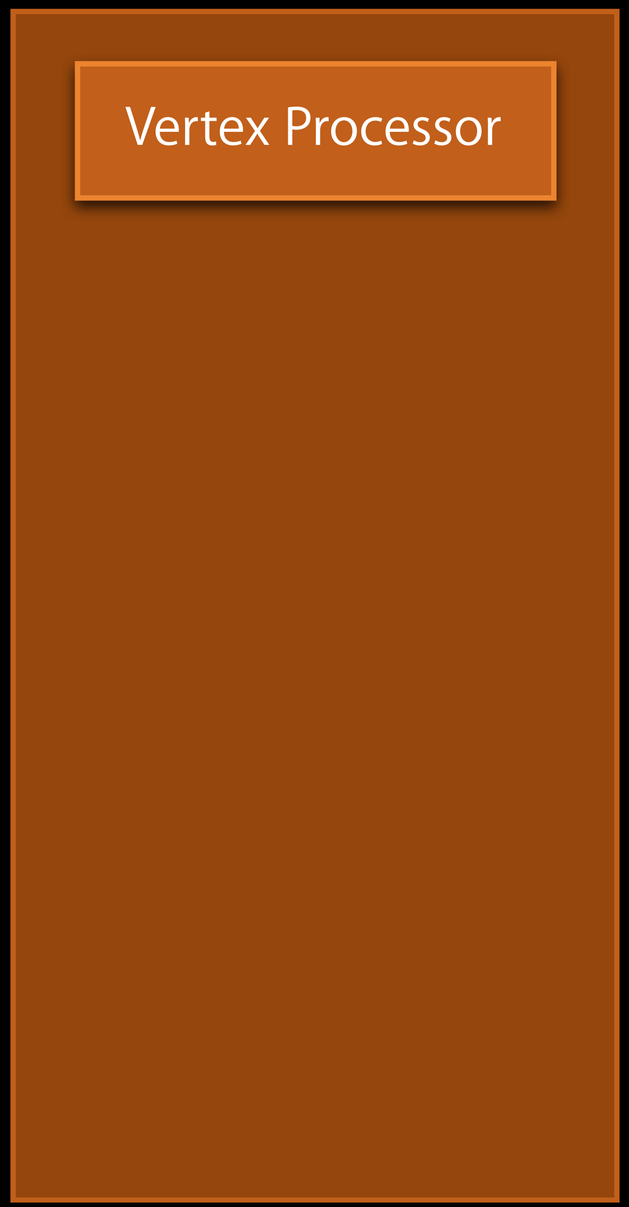
High performance, low power

- TBDR Pipeline different than traditional streaming GPUs
- Optimizations to reduce processing load
  - Increases performance
  - Saves power
- Depends heavily on caches
  - Large transfers to unified memory costly
- Certain operations prevent optimizations or cause cache misses
  - Operations avoidable

The Tile-Based Deferred Rendering Pipeline

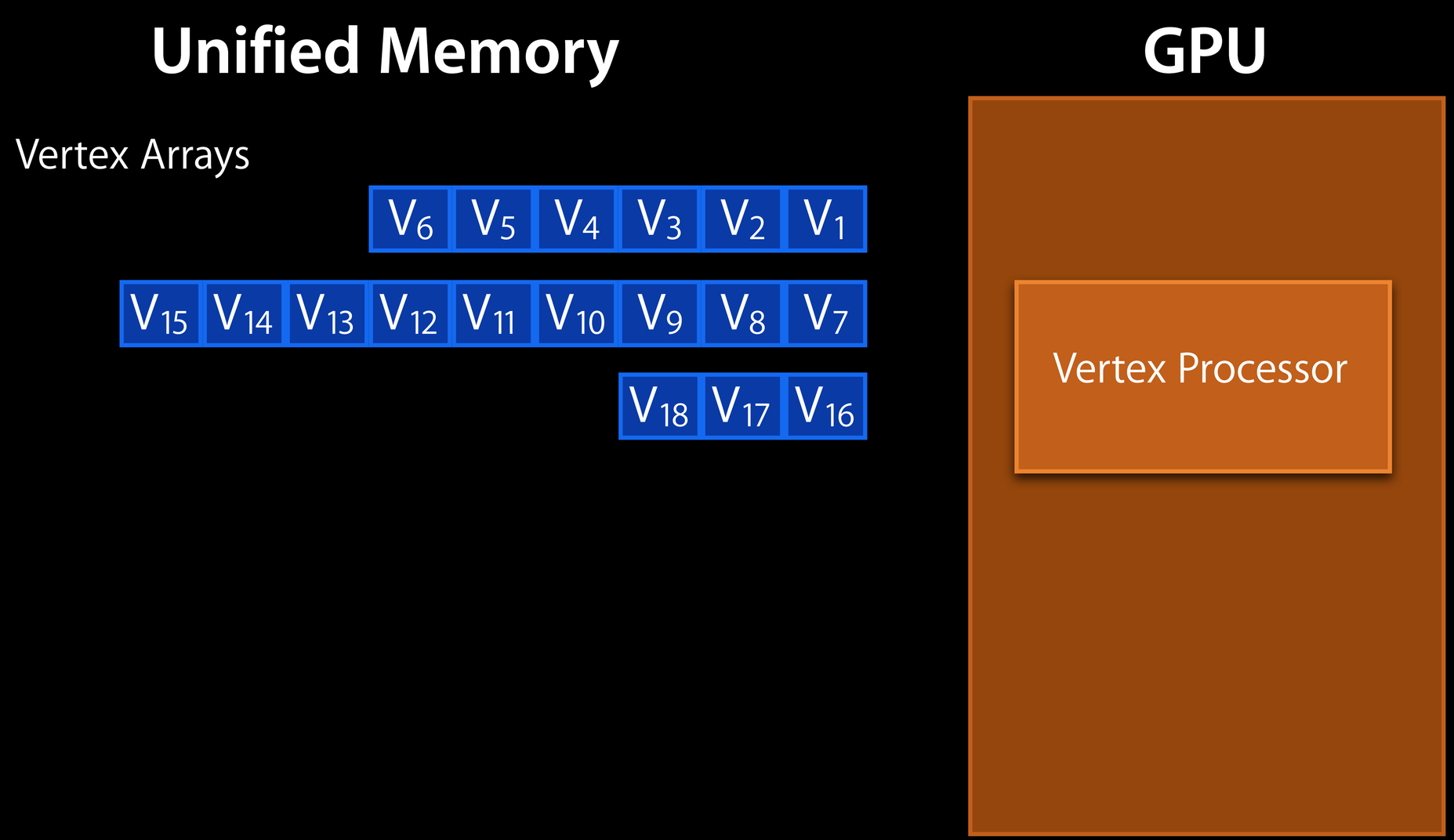


The Tile-Based Deferred Rendering Pipeline



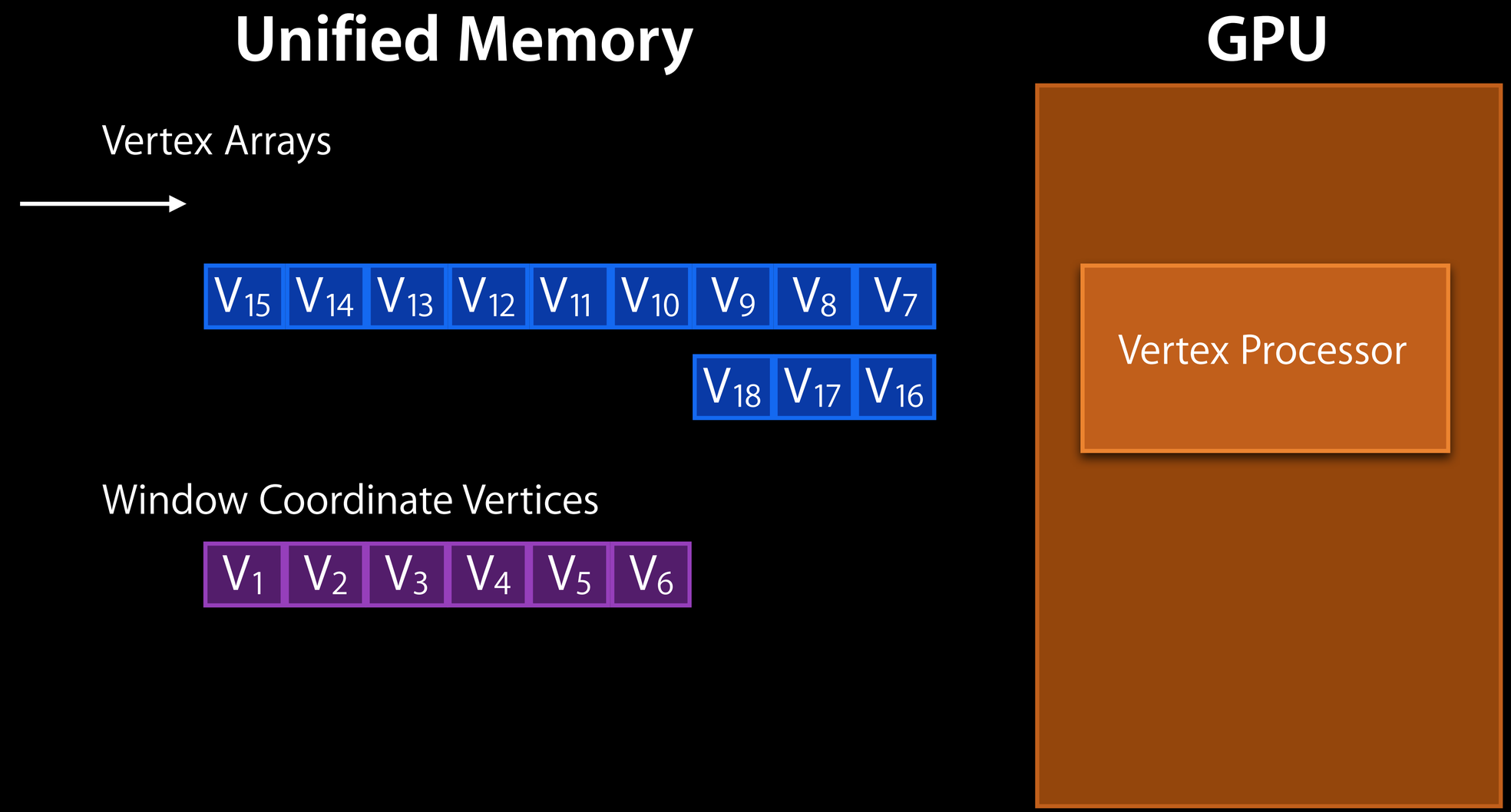
Vertex Shading

- Draw call submits vertices to GPU's vertex processor

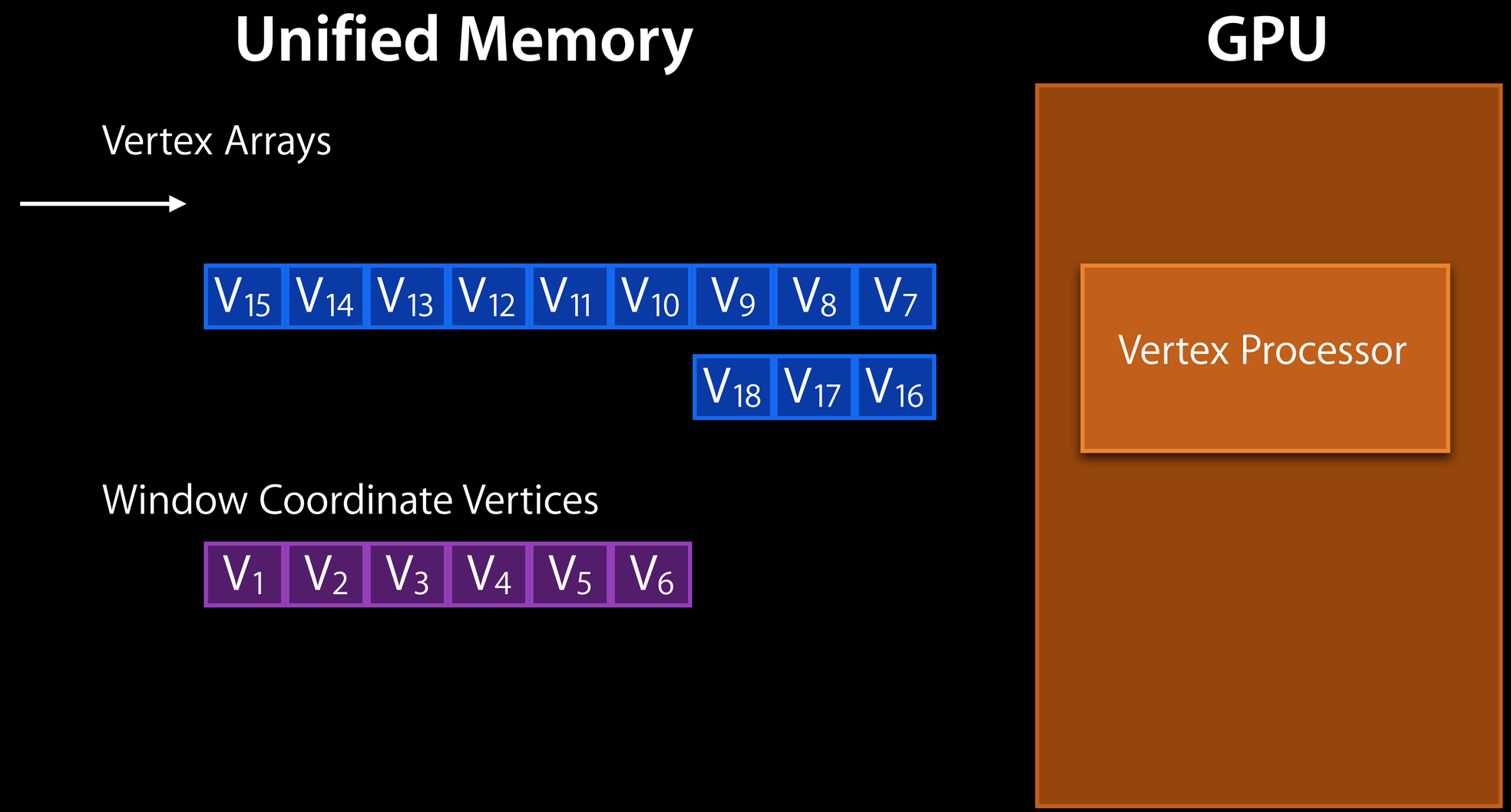


Vertex Shading

- Draw call submits vertices to GPU's vertex processor



- Vertices shaded and transformed to window coords
- Shaded, transform vertices, stored out to unified memory



- Vertices shaded and transformed to window coords
- Shaded, transform vertices, stored out to unified memory

Unified Memory

Vertex Arrays

Window Coordinate Vertices

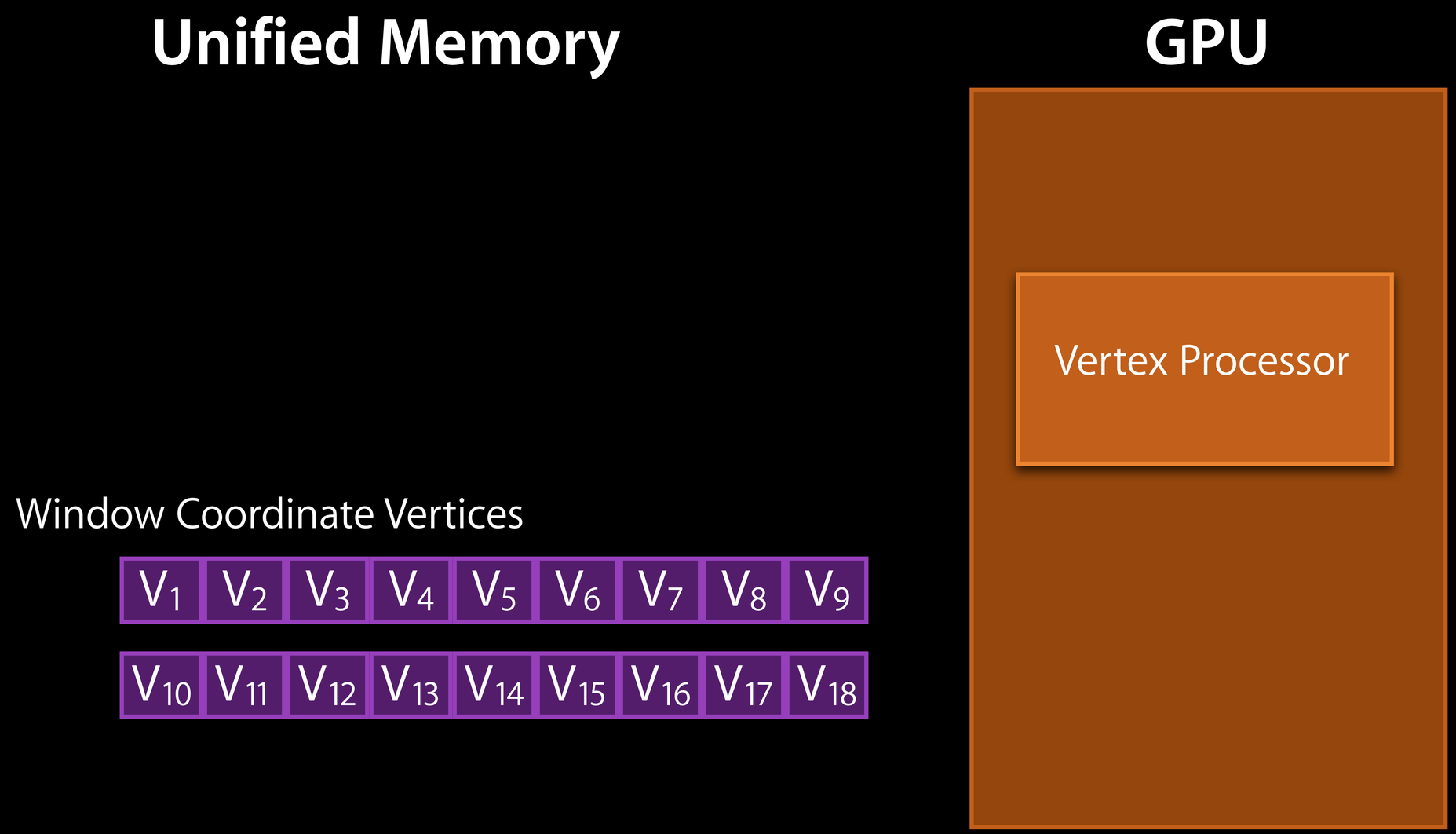
V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9

V_{10}	V_{11}	V_{12}	V_{13}	V_{14}	V_{15}	V_{16}	V_{17}	V_{18}

GPU

Vertex Processor

- A frames' worth of vertices stored
- Rasterization deferred until `presentRenderbuffer` or renderbuffer changed



Tiling Processor

Unified Memory

Window Coordinate Vertices

V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9

V_{10}	V_{11}	V_{12}	V_{13}	V_{14}	V_{15}	V_{16}	V_{17}	V_{18}

GPU

Vertex Processor

Tiling Processor

Unified Memory

Window Coordinate Vertices

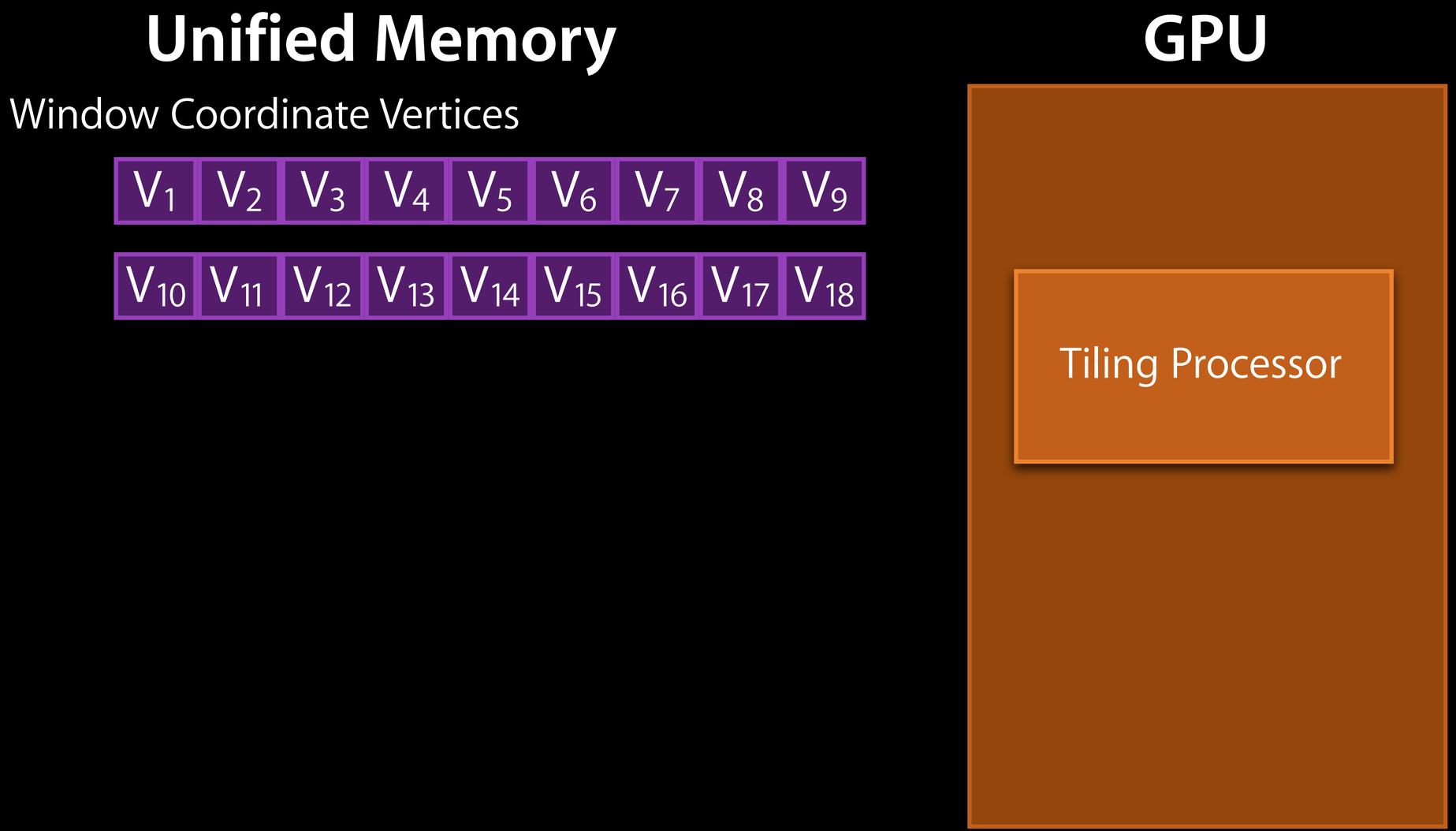
V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9

V_{10}	V_{11}	V_{12}	V_{13}	V_{14}	V_{15}	V_{16}	V_{17}	V_{18}

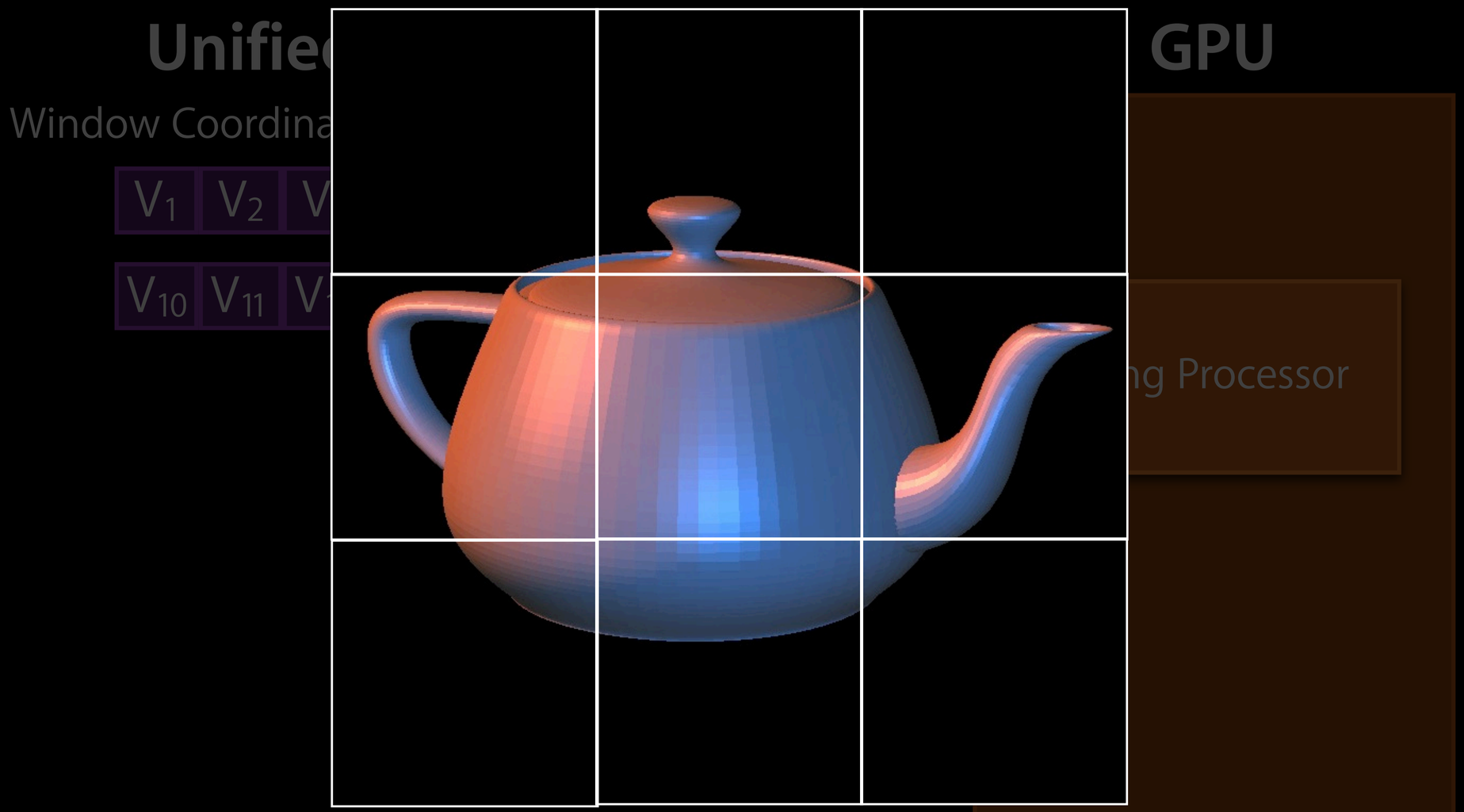
GPU

Tiling Processor

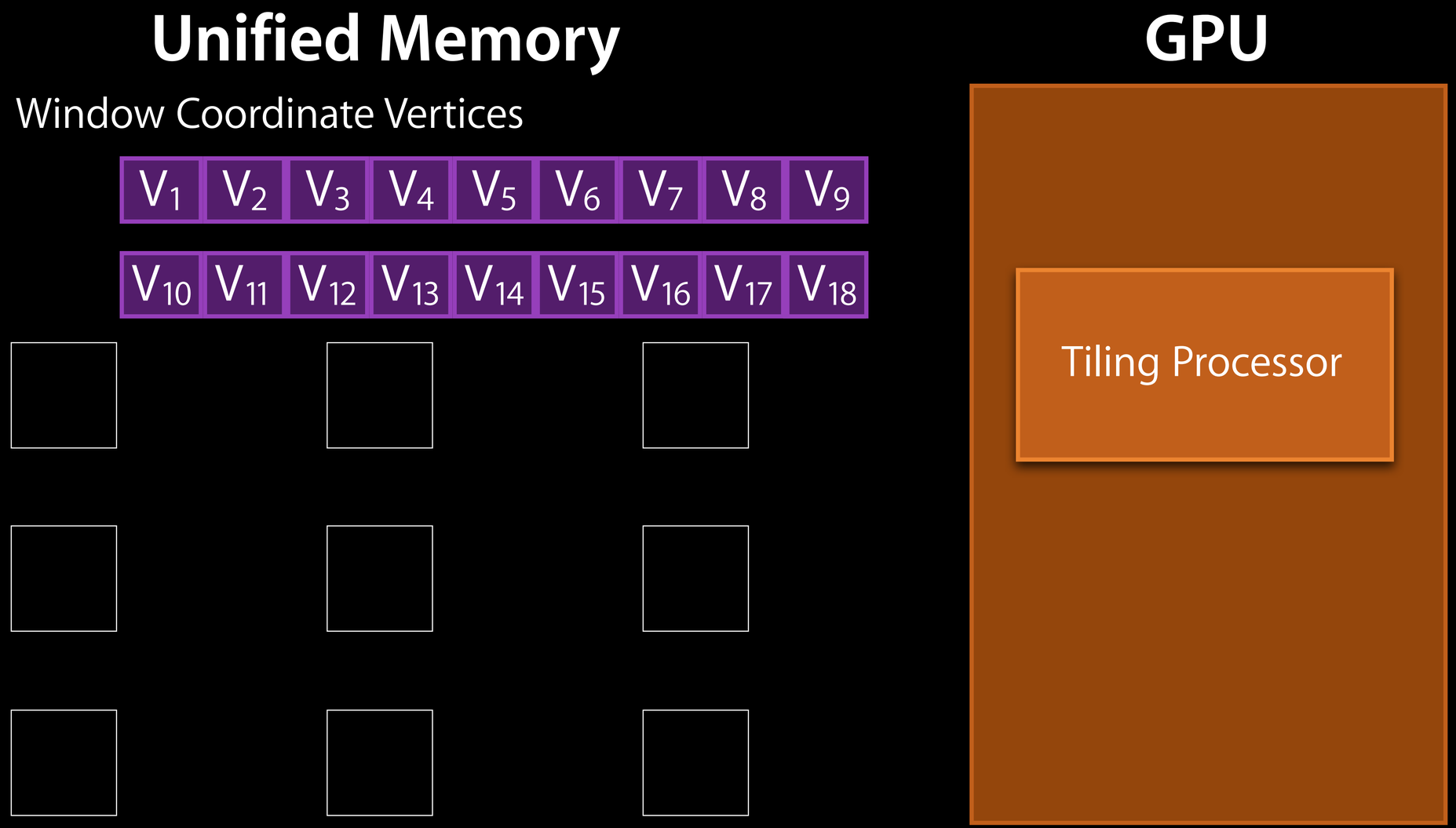
- Render buffers split into tiles
- Allows rasterization and fragment shading on embedded GPU memory



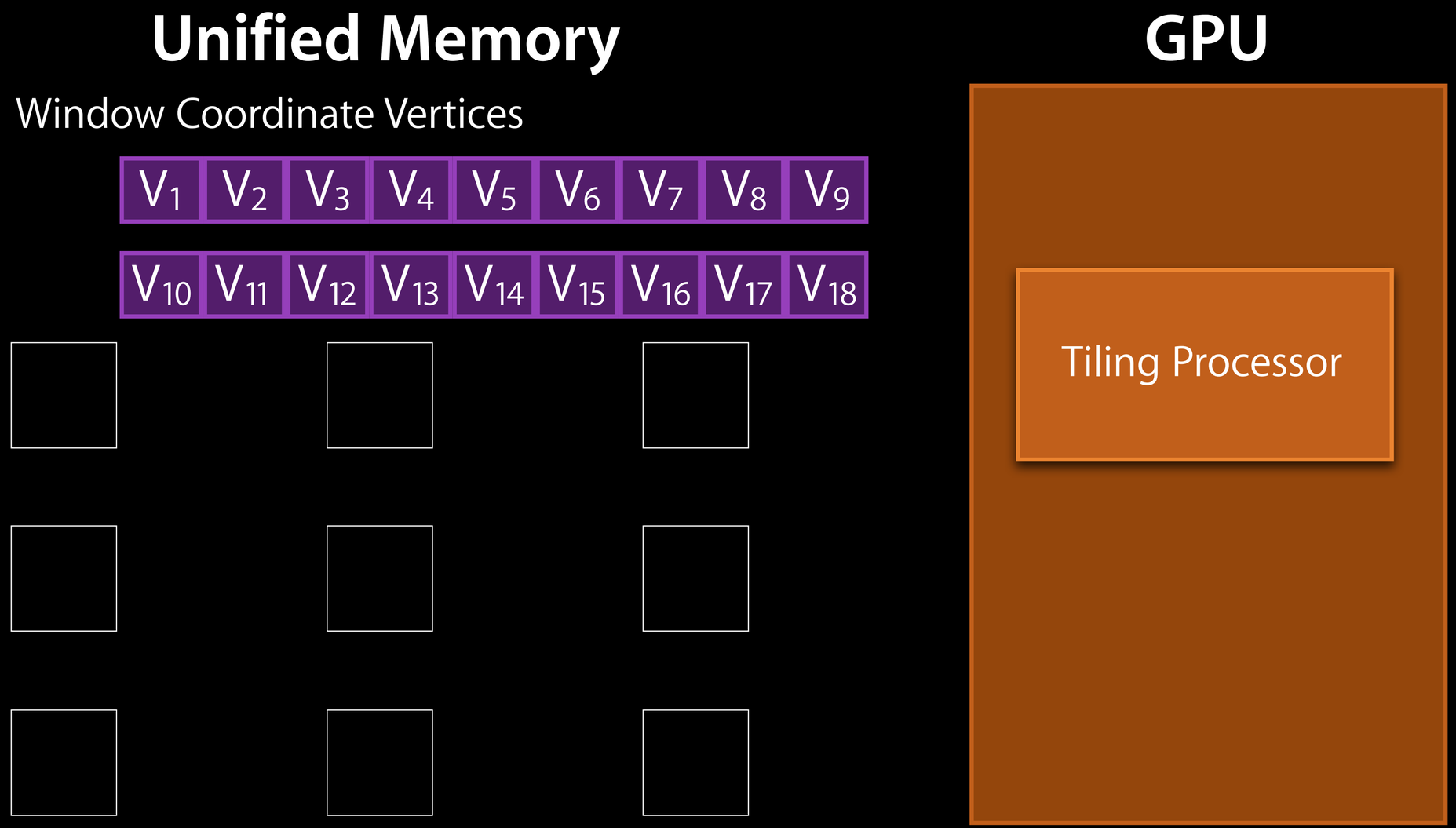
- Render buffers split into tiles
- Allows rasterization and fragment shading on embedded GPU memory



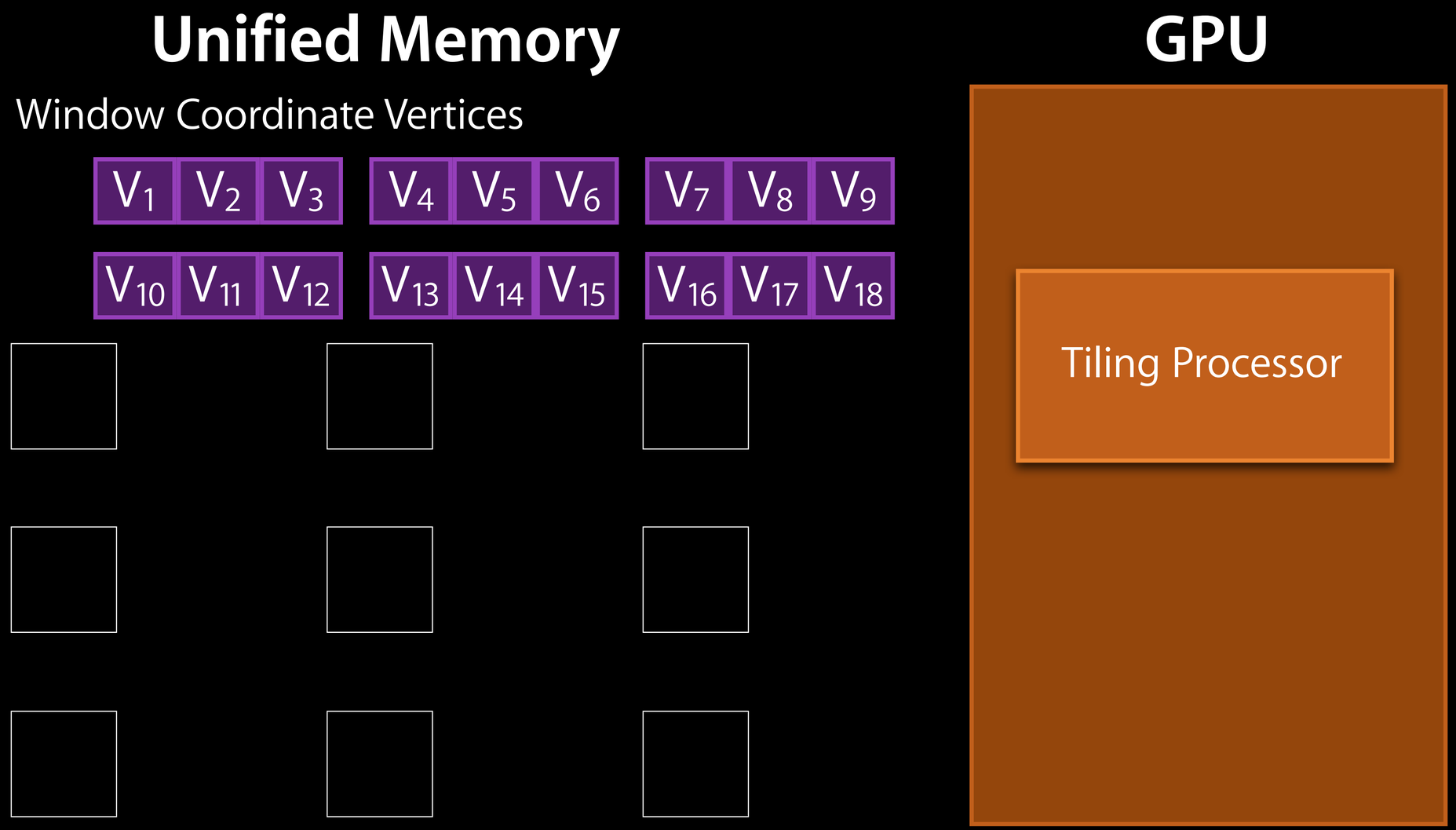
- Render buffers split into tiles
- Allows rasterization and fragment shading on embedded GPU memory



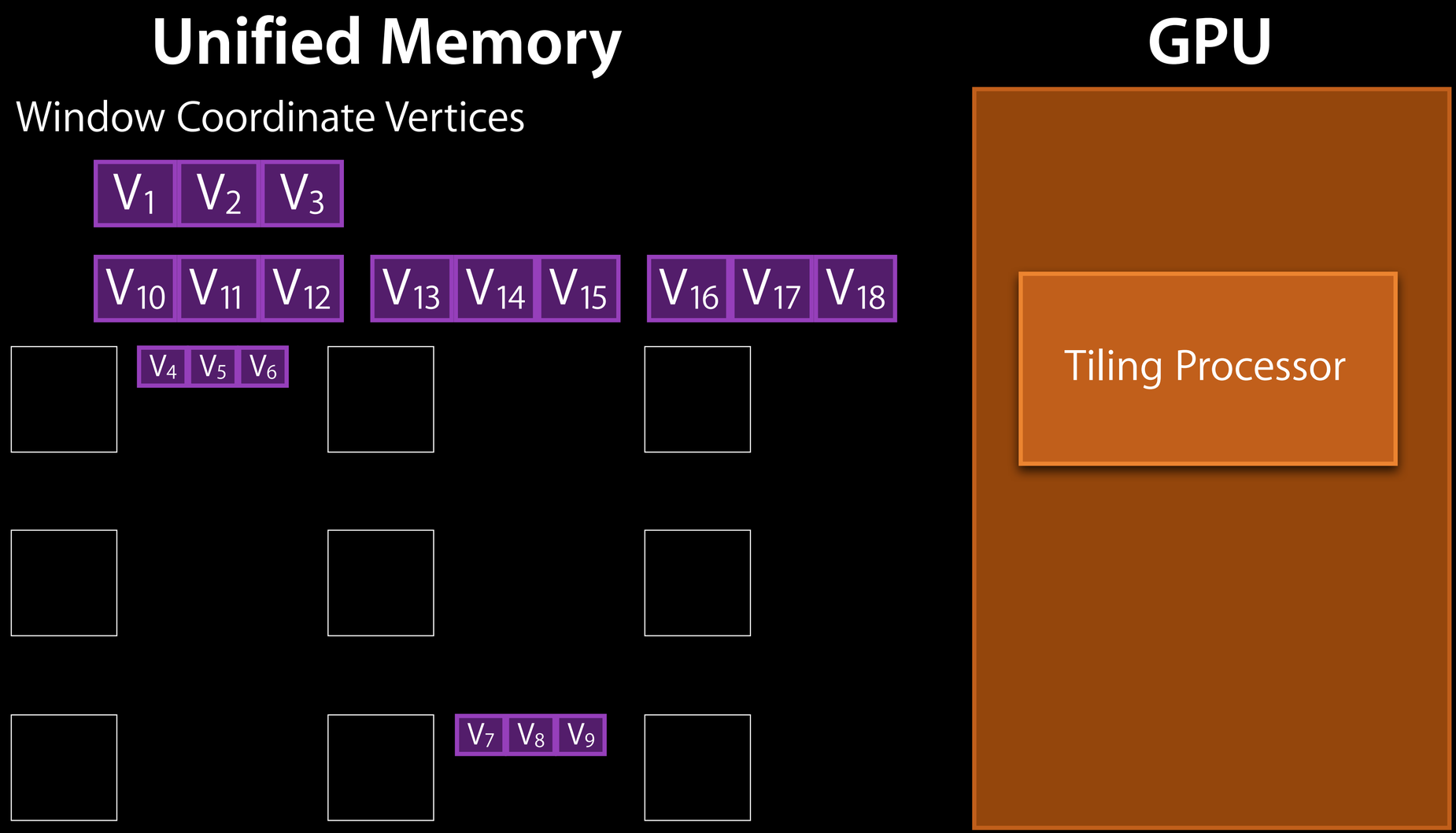
- Vertices processed in groups of triangles
- Tiling processor bins triangles into tiles in which they are located



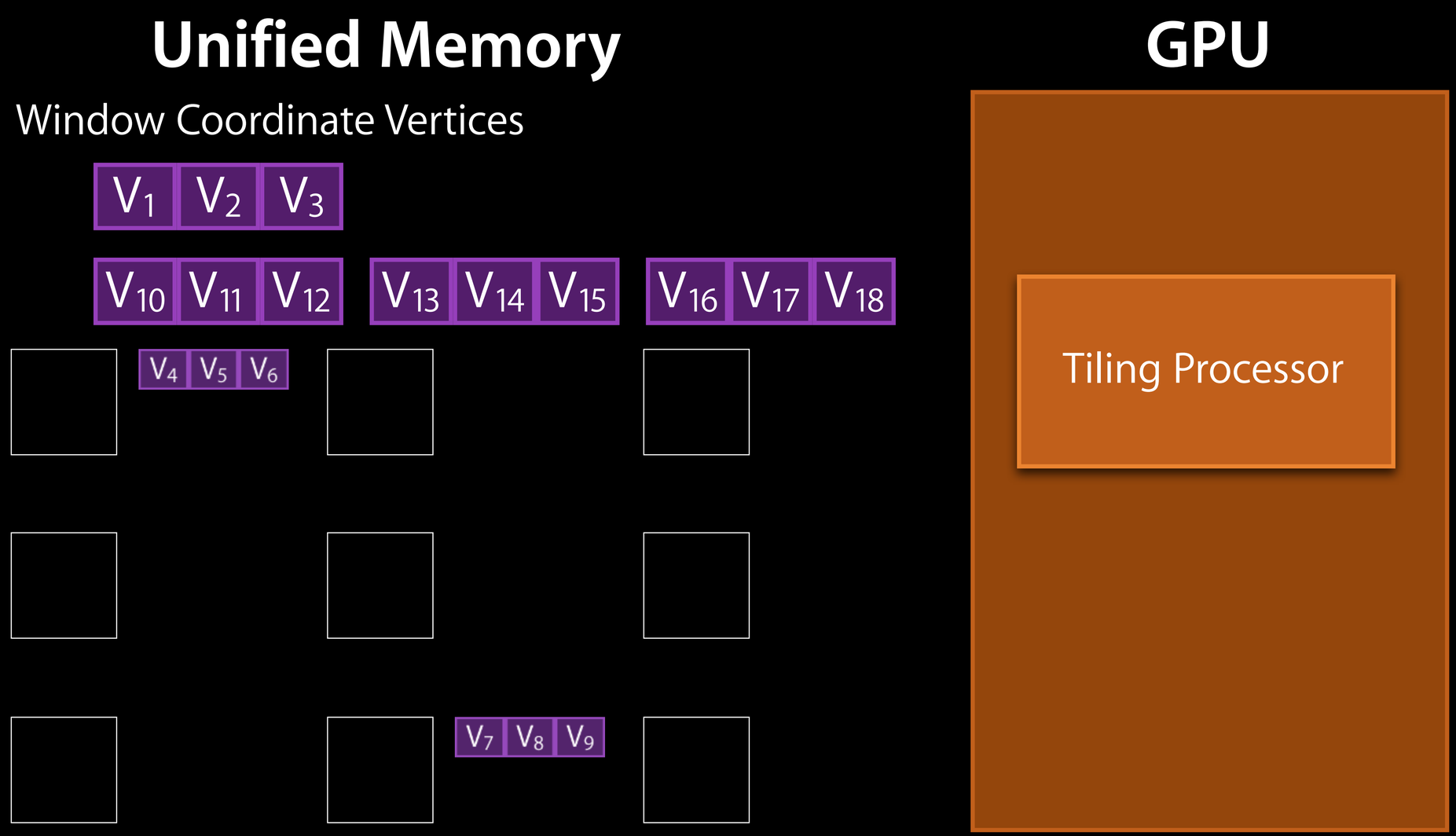
- Vertices processed in groups of triangles
- Tiling processor bins triangles into tiles in which they are located



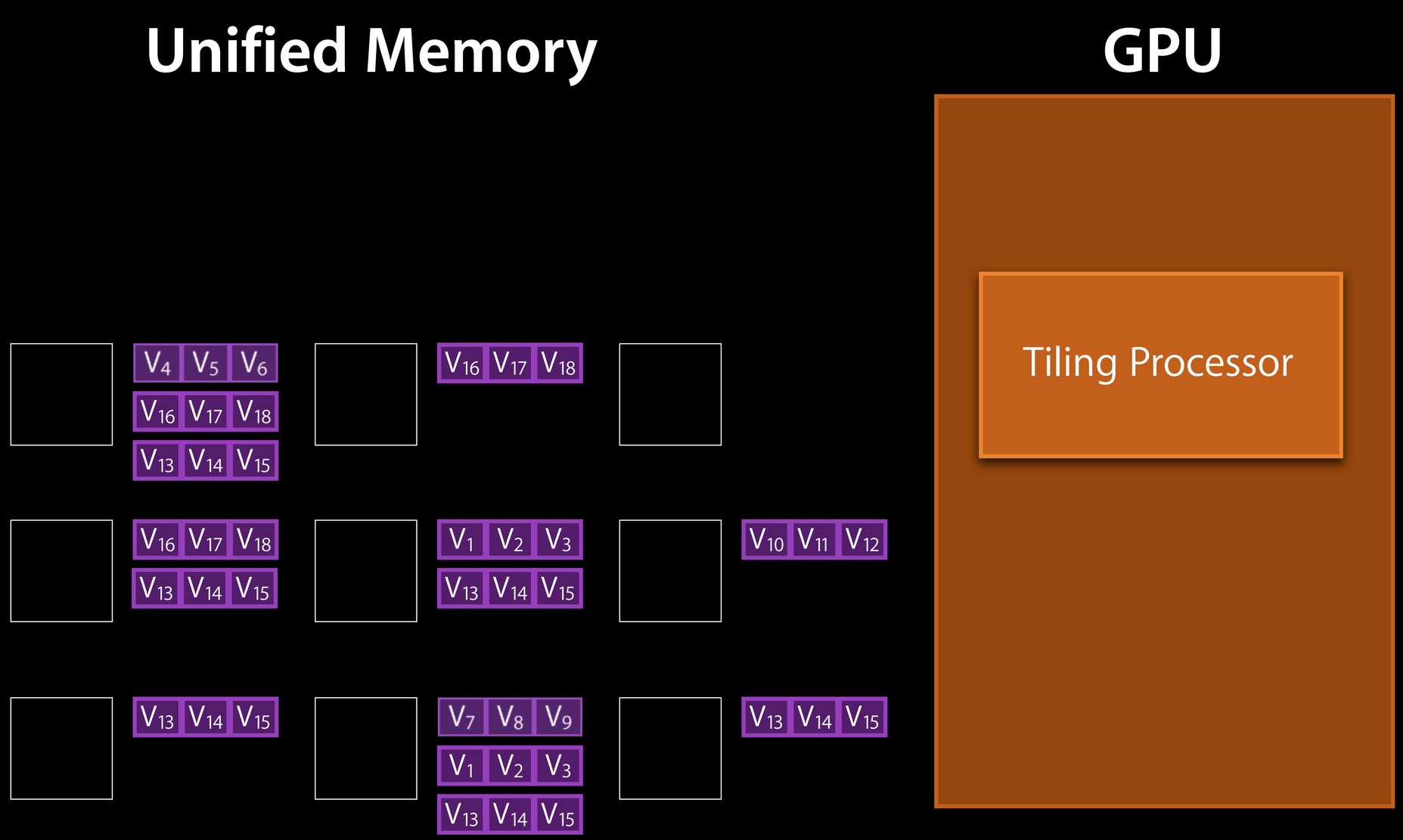
- Vertices processed in groups of triangles
- Tiling processor bins triangles into tiles in which they are located



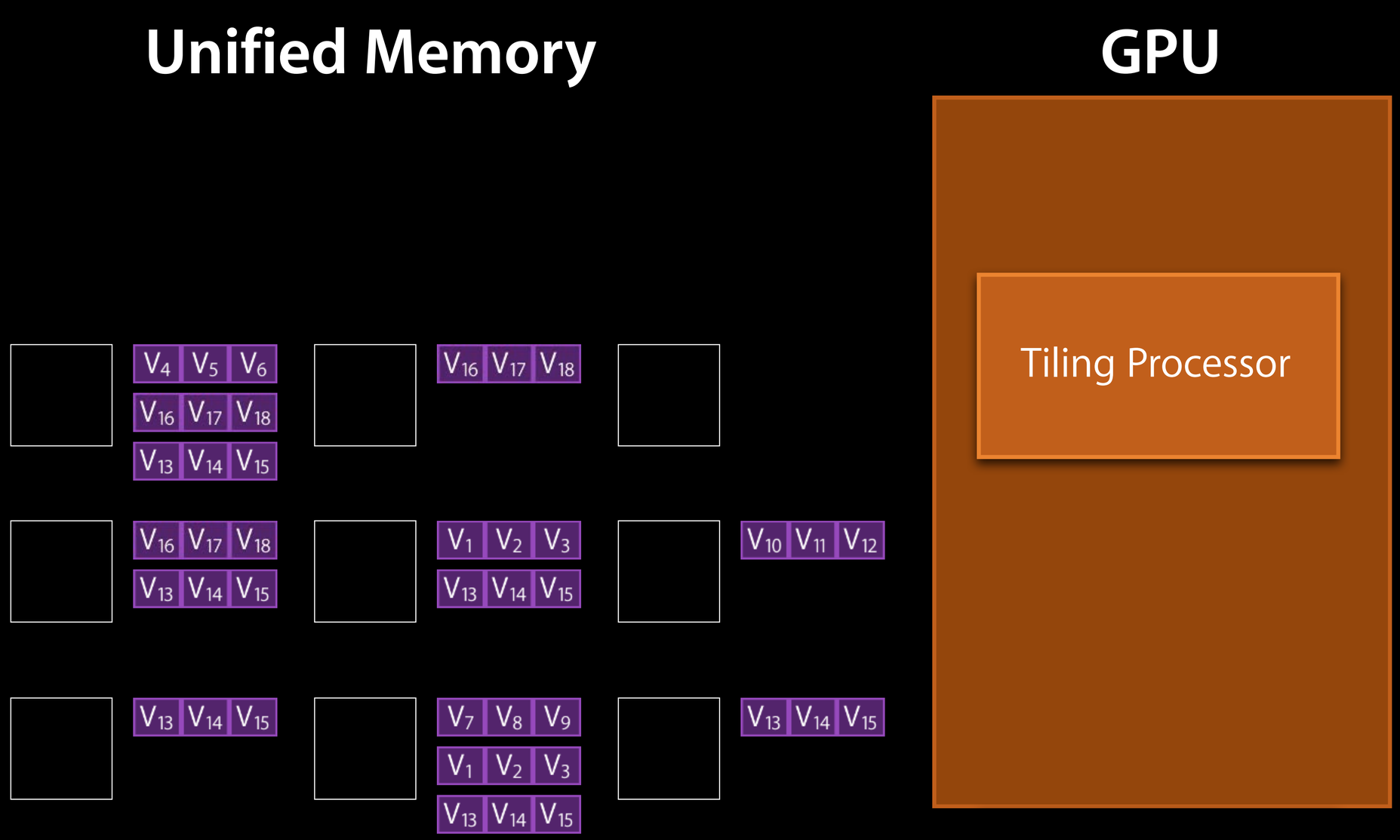
- GPU may bin a larger triangle into multiple tiles



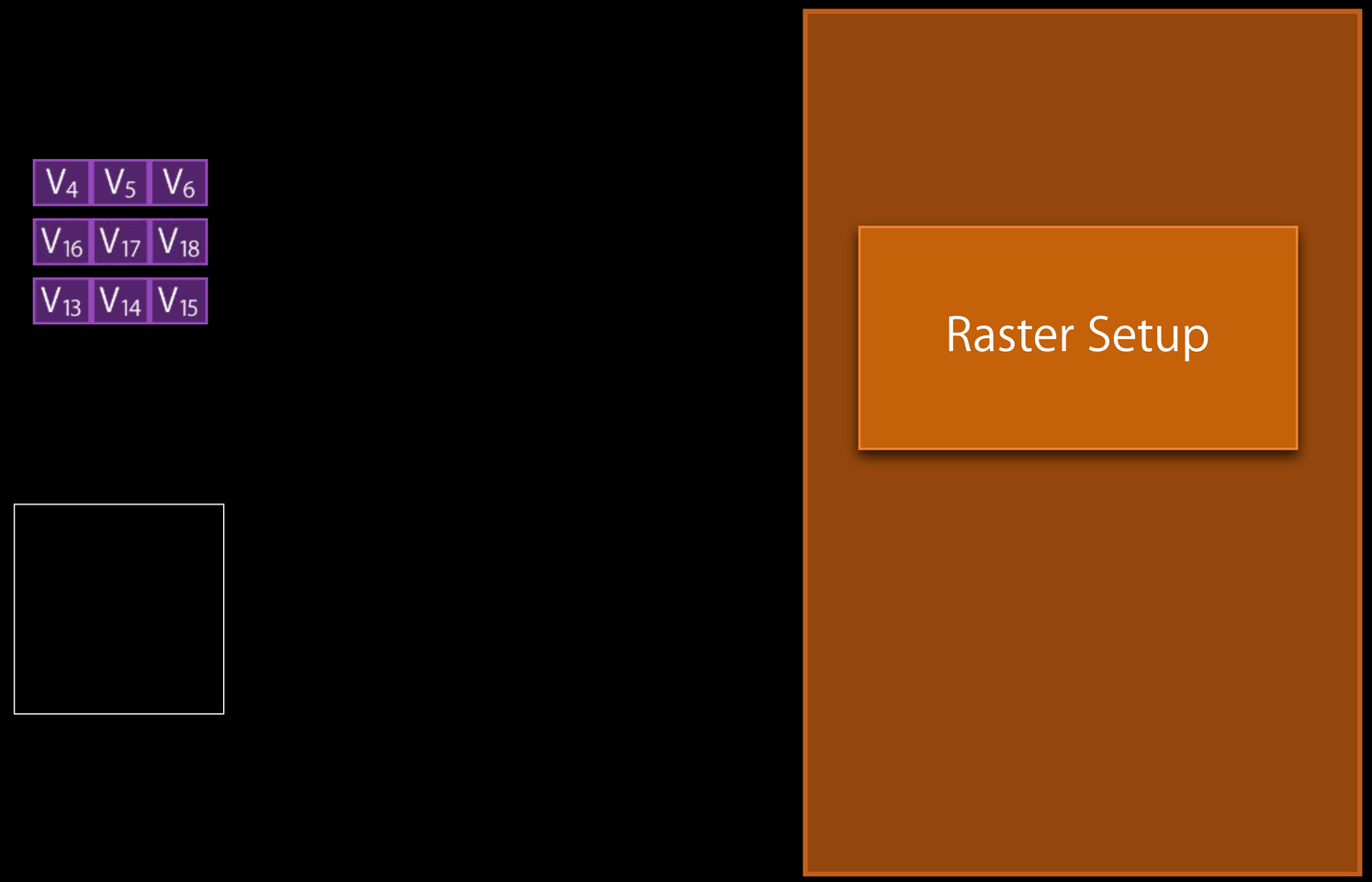
- GPU may bin a larger triangle into multiple tiles



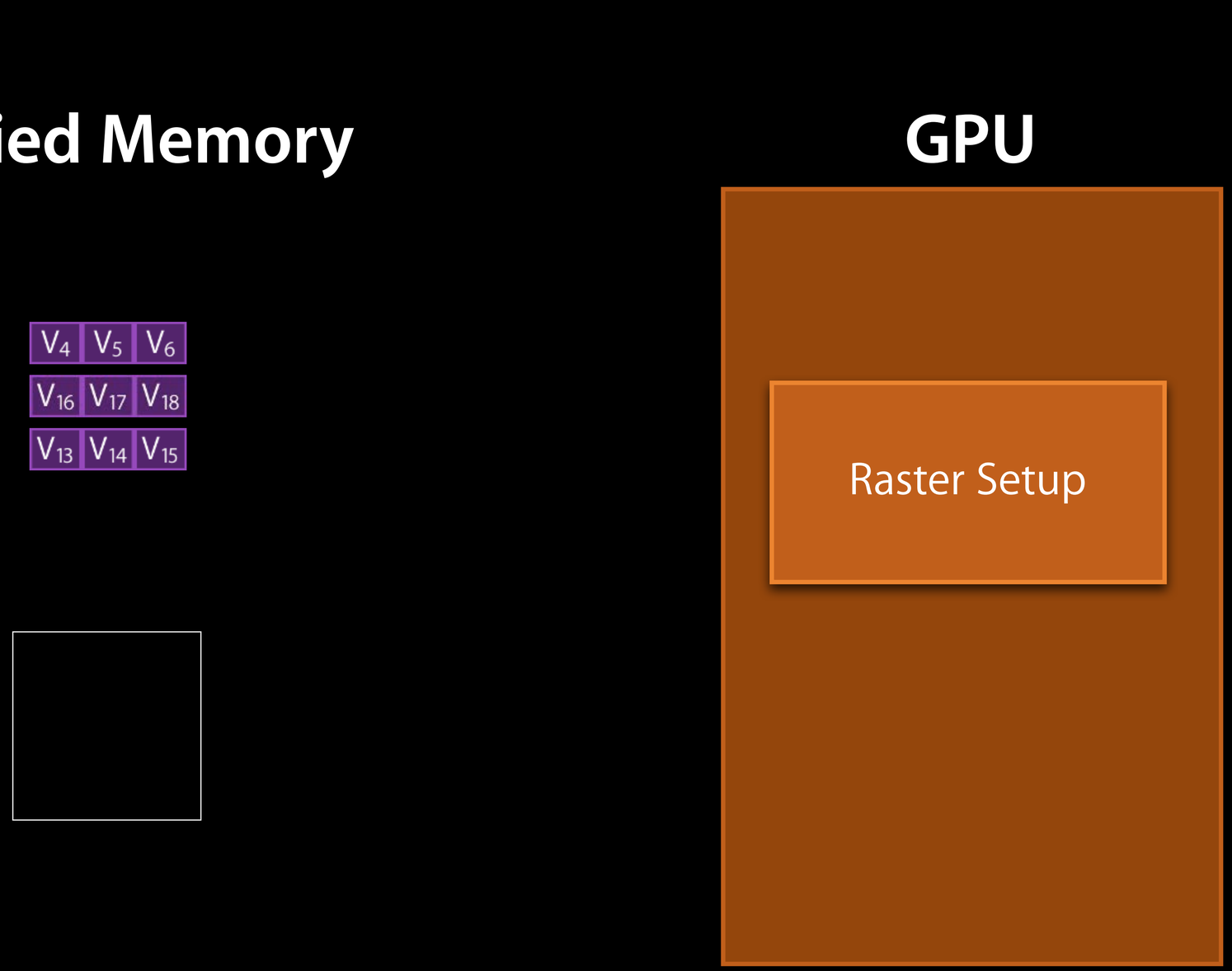
Raster Setup



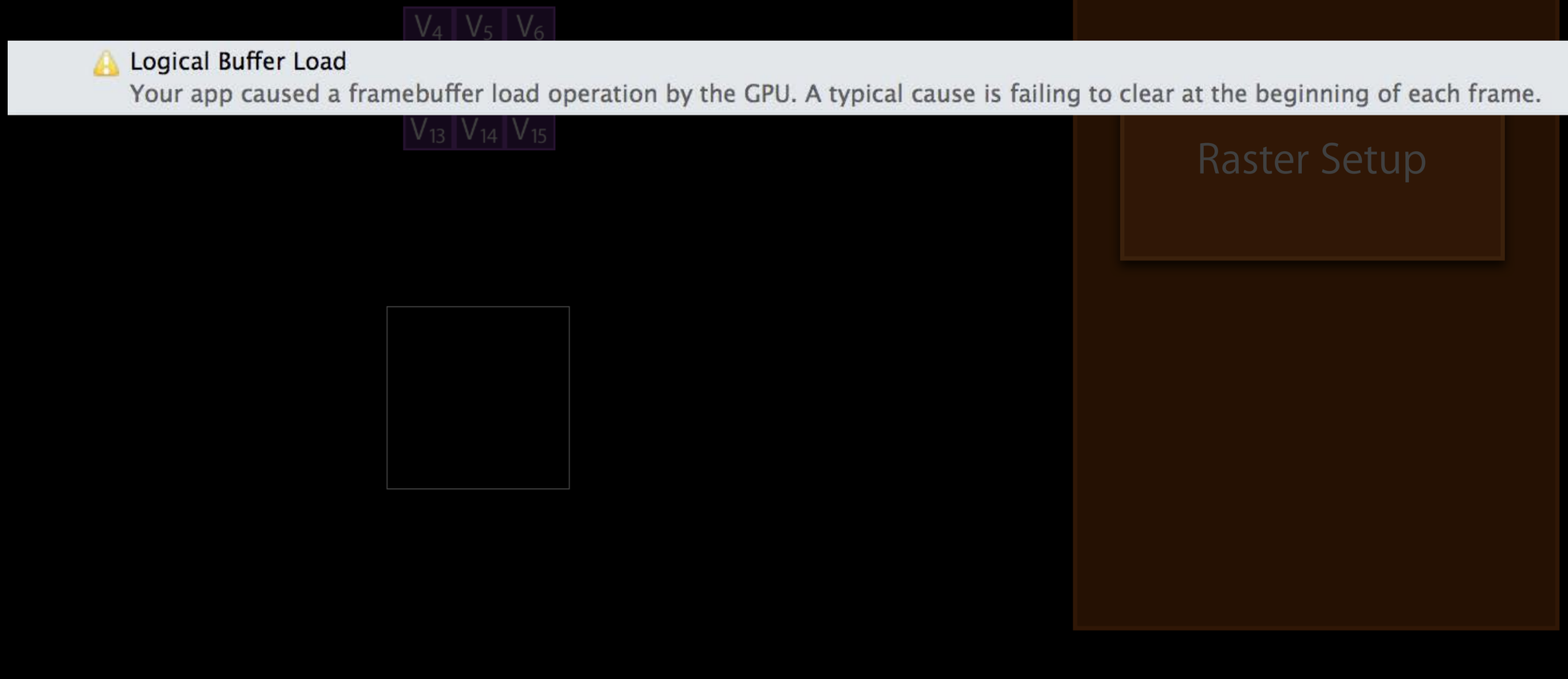
Raster Setup



Raster Setup



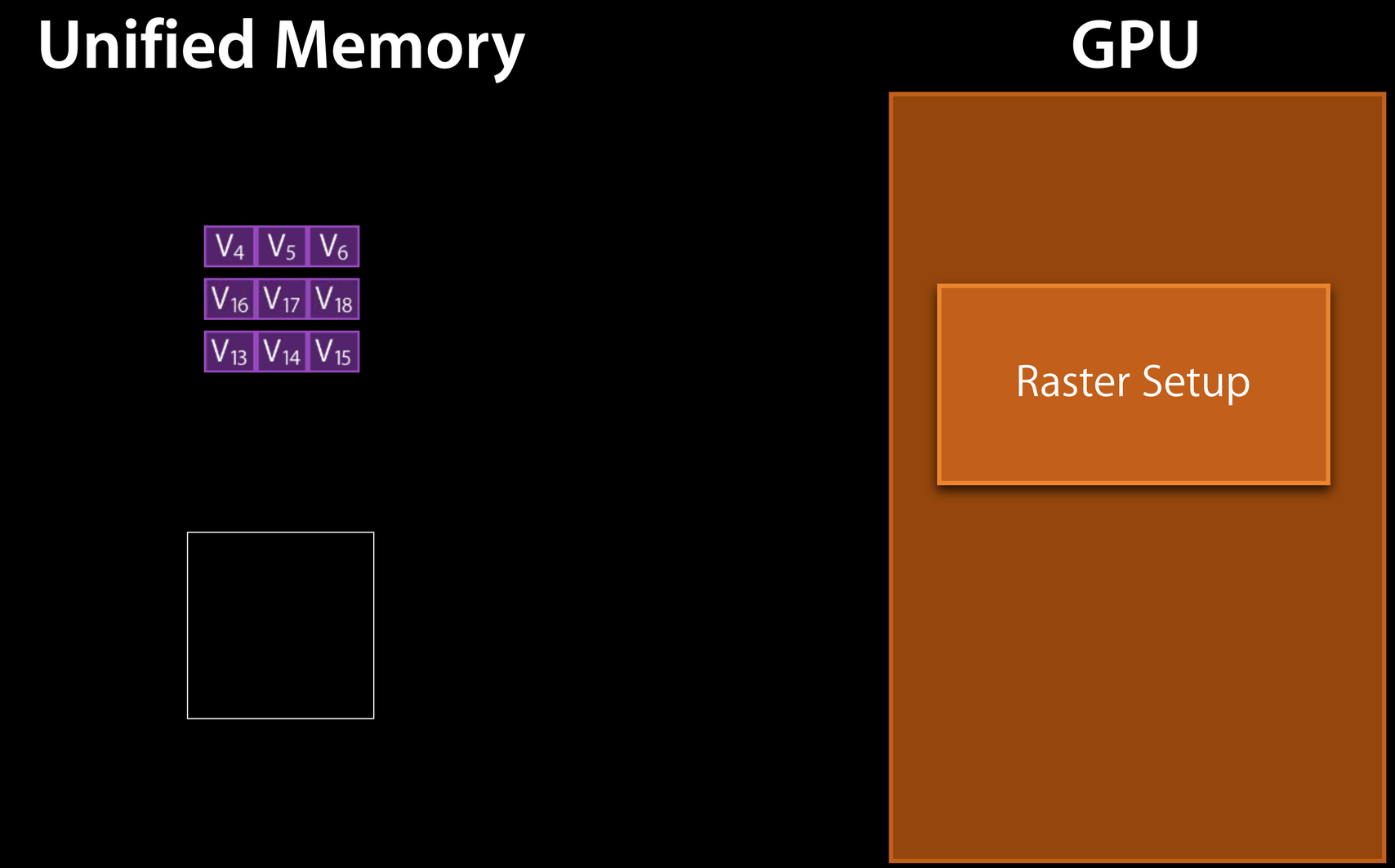
Raster Setup



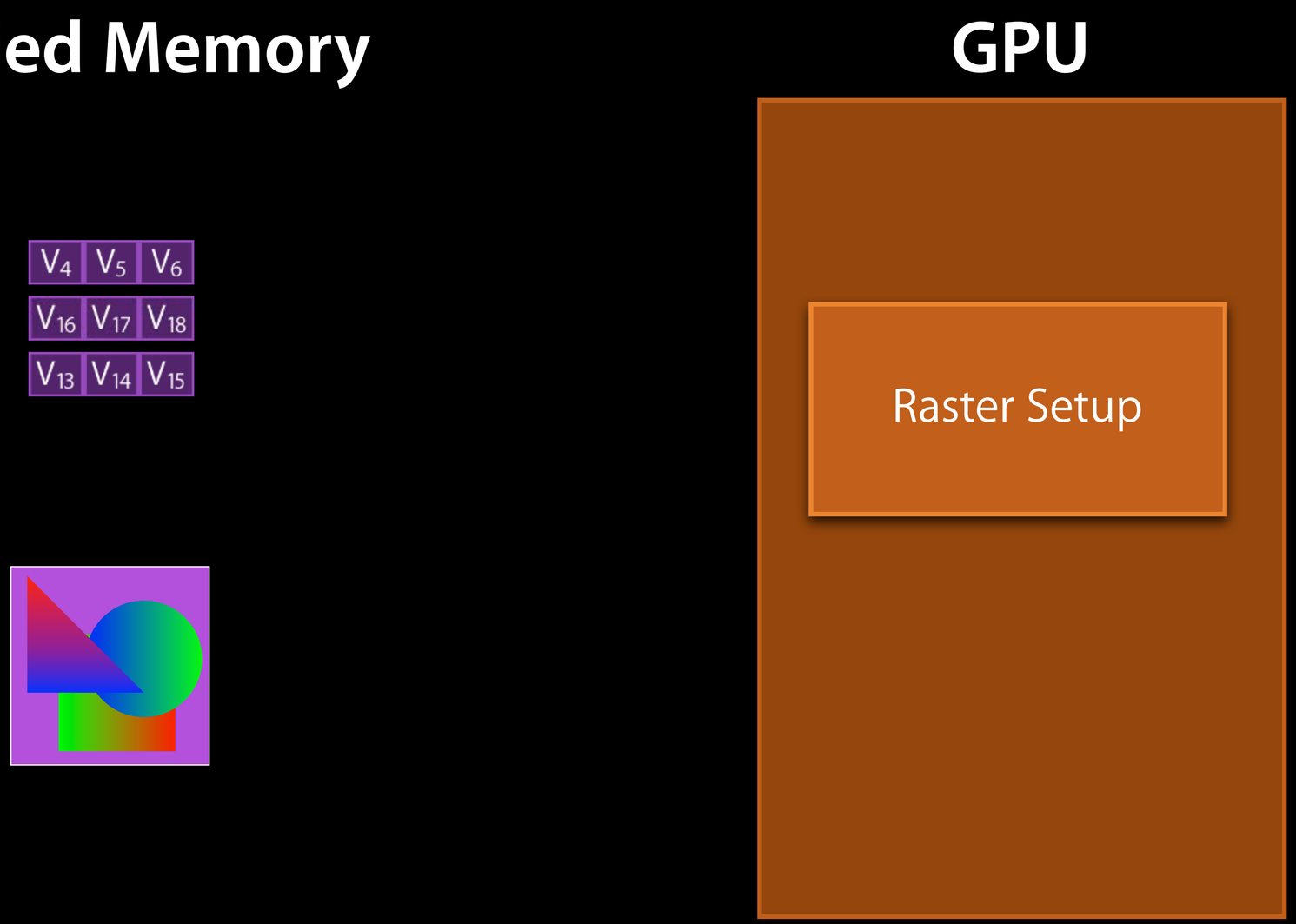
Logical Buffer Load

Your app caused a framebuffer load operation by the GPU. A typical cause is failing to clear at the beginning of each frame.

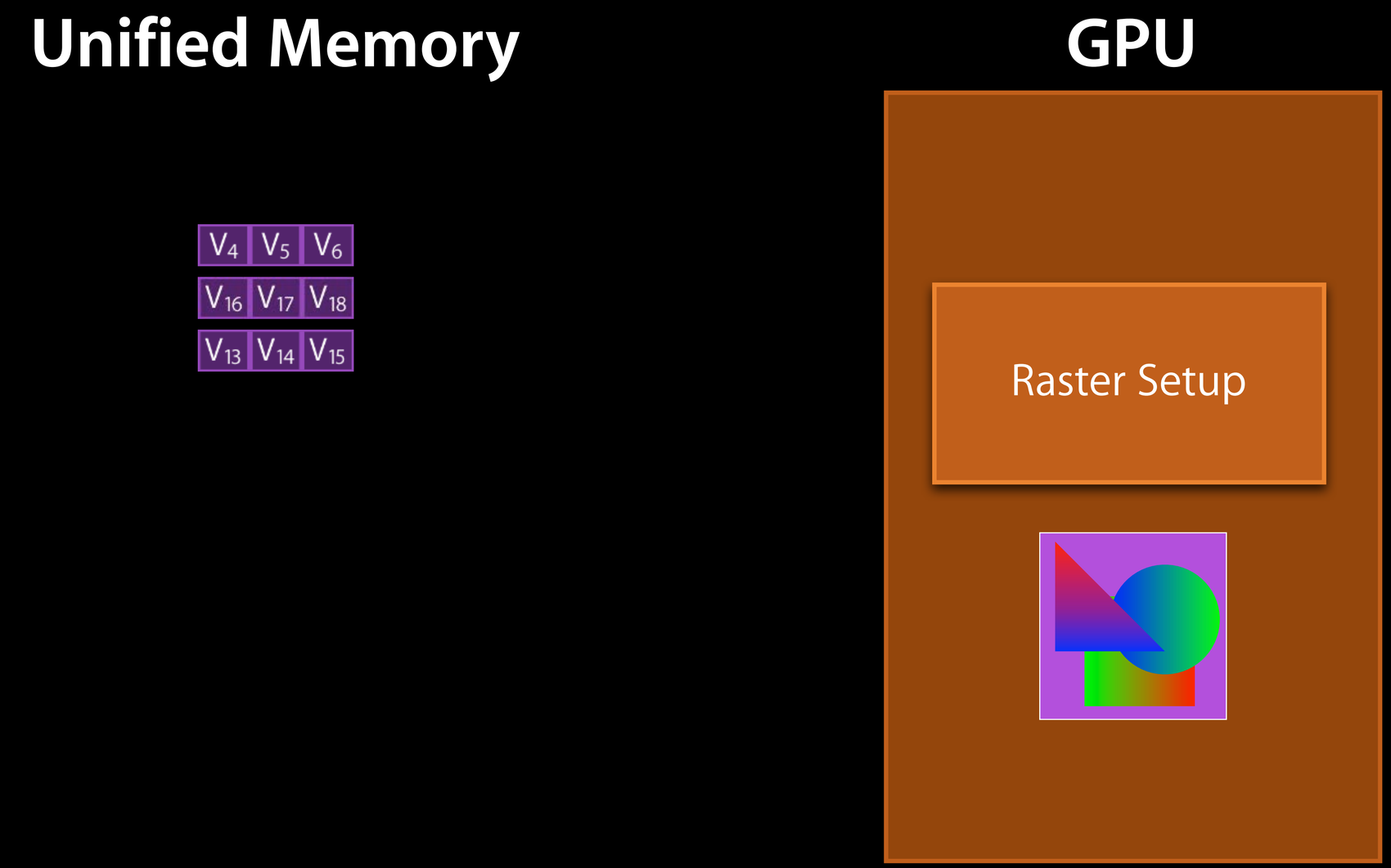
- Rasterizer uses tile-sized embedded memory



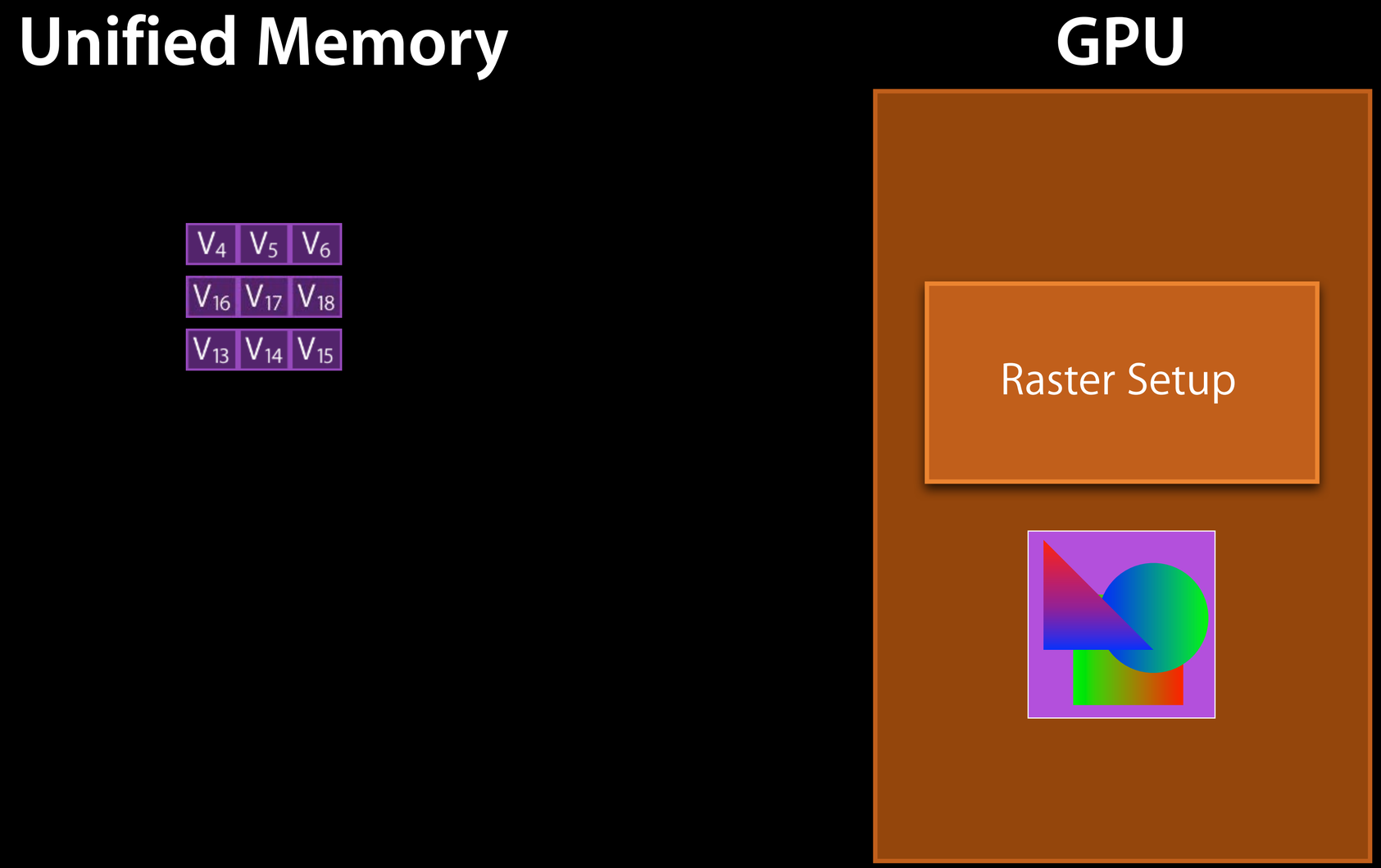
- Rasterizer uses tile-sized embedded memory
- If data is in render buffer, GPU must perform a costly load



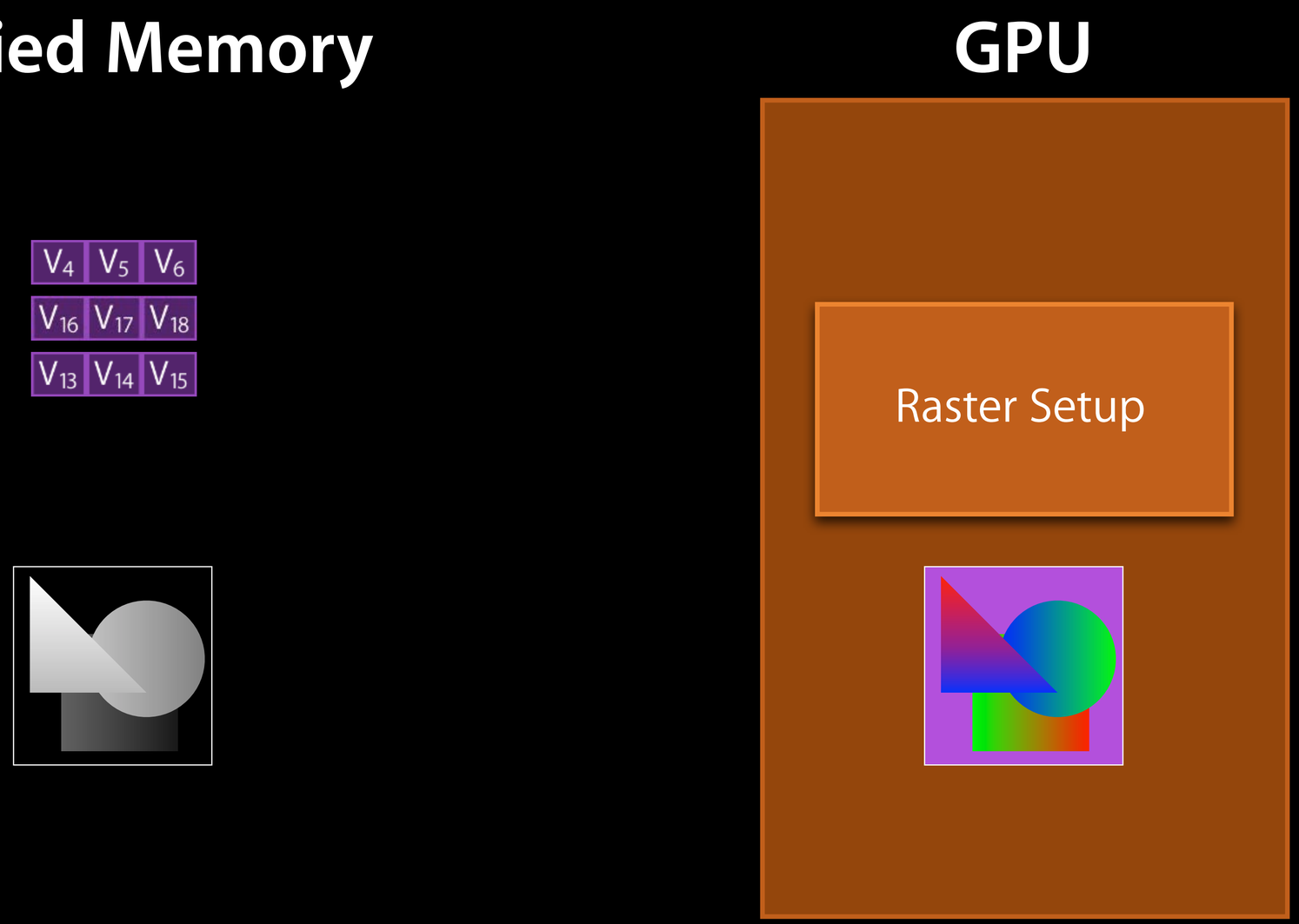
- Rasterizer uses tile-sized embedded memory
- If data is in render buffer, GPU must perform a costly load



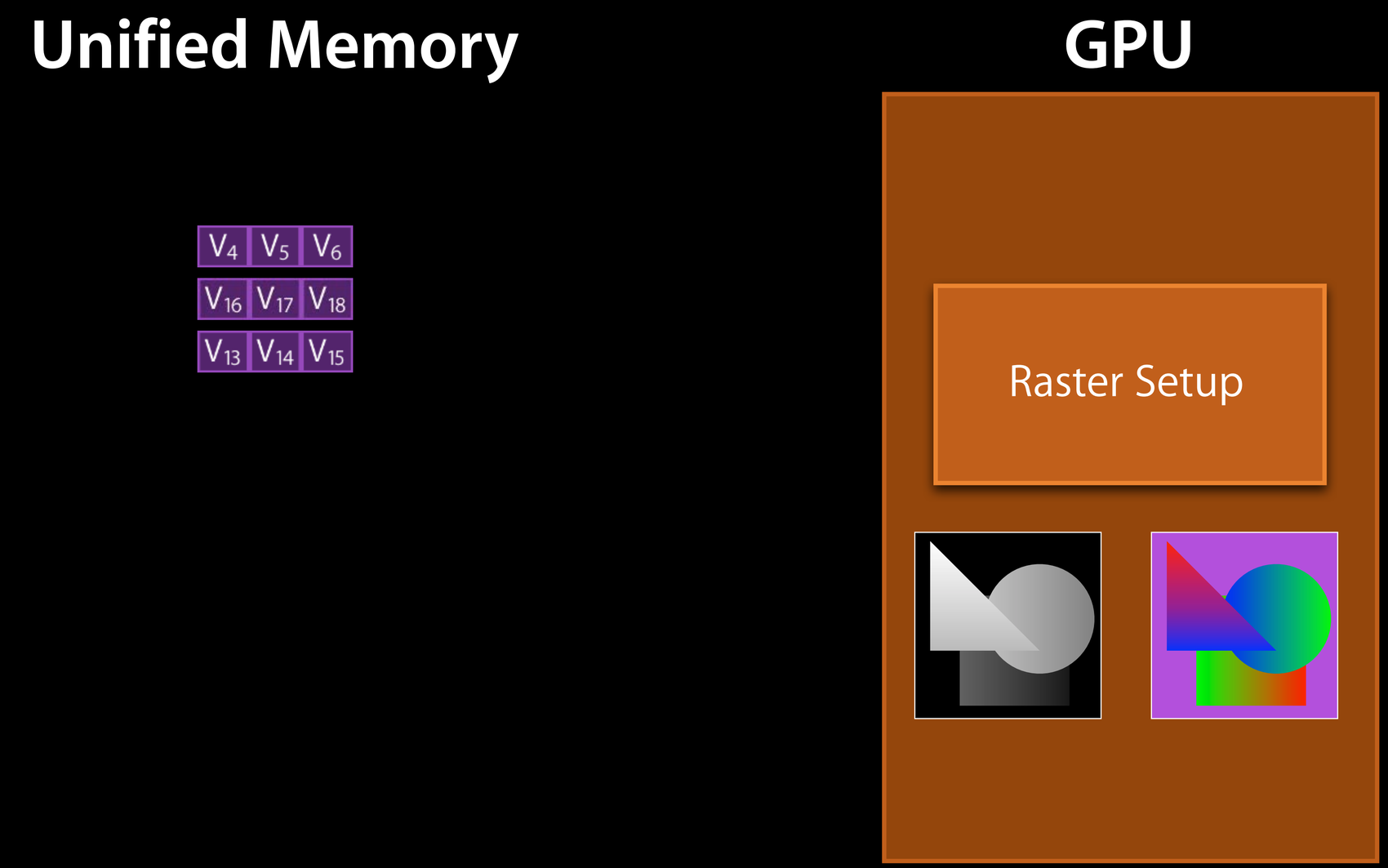
- If there is data in the depth buffer, GPU must load it also



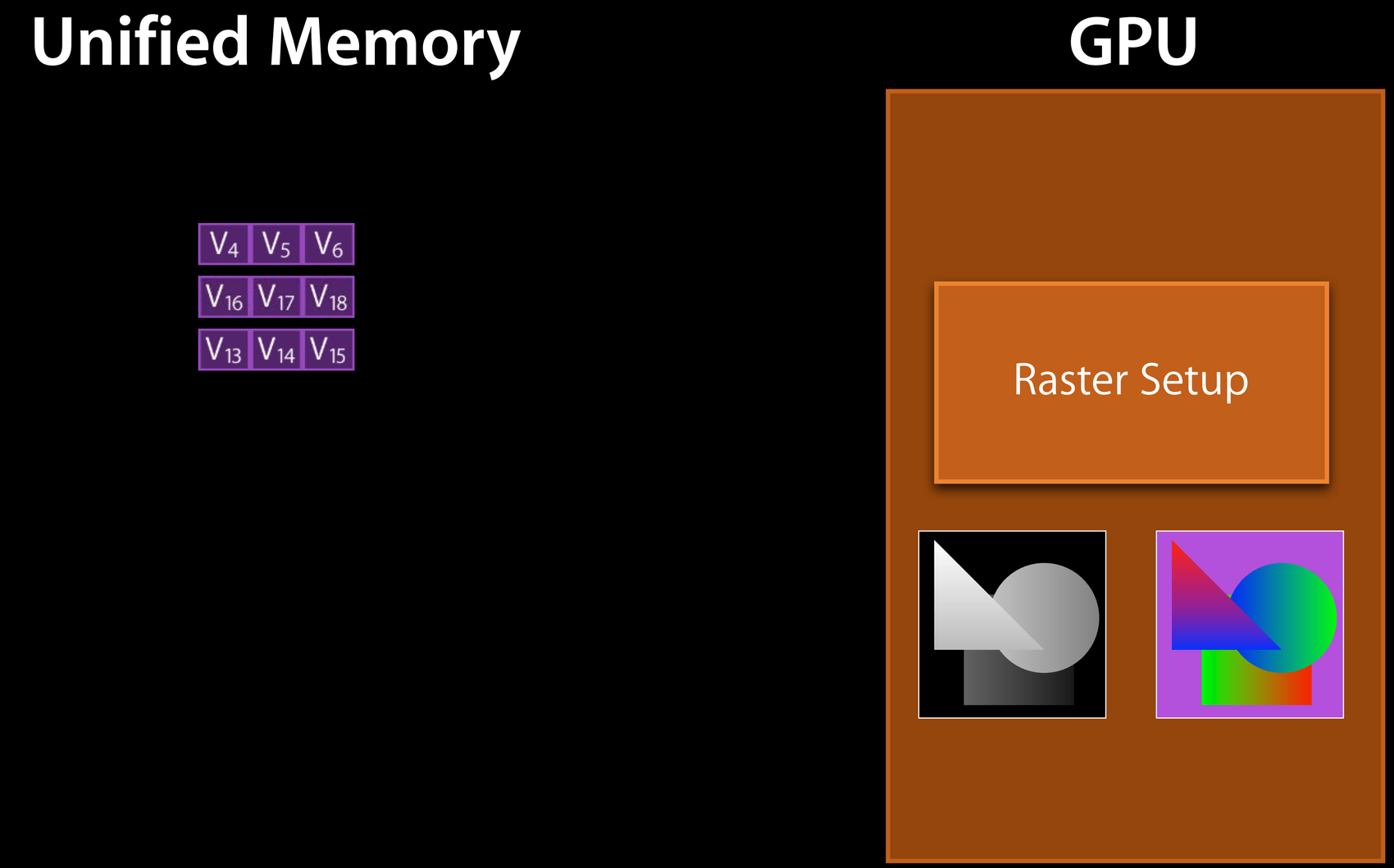
- If there is data in the depth buffer, GPU must load it also



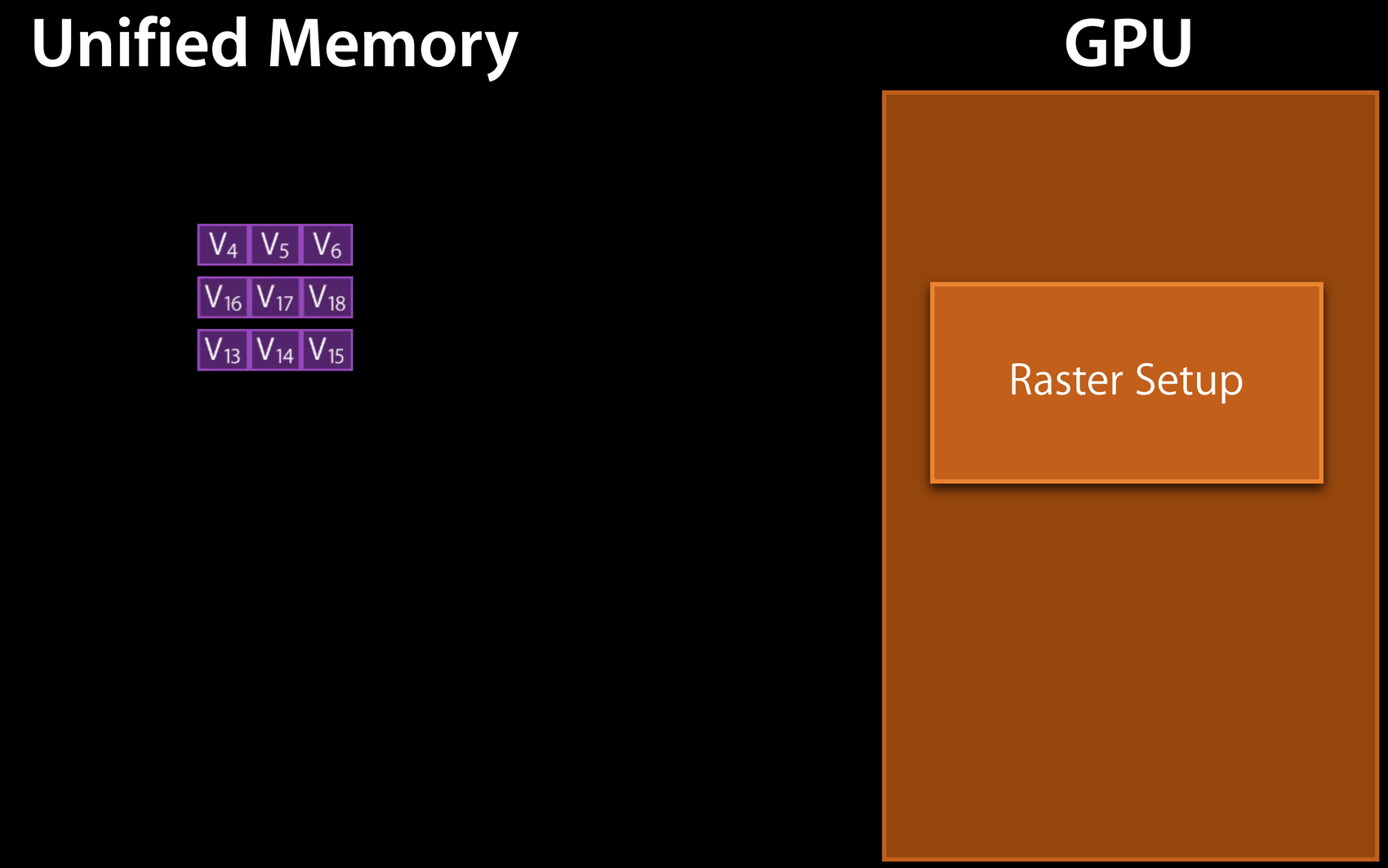
- If there is data in the depth buffer, GPU must load it also



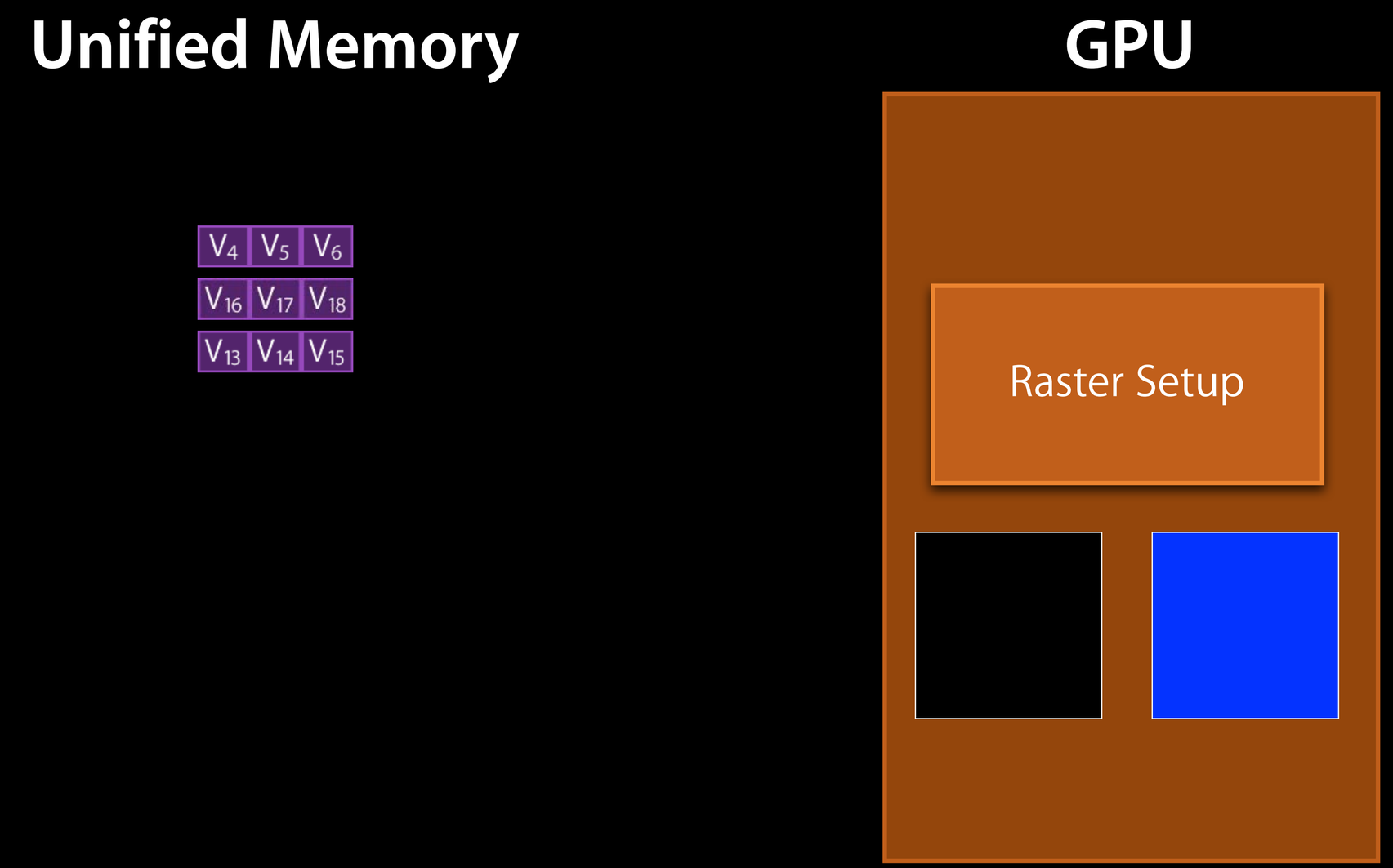
- Loading a tile is called a Logical Buffer Load
  - Developers can avoid these



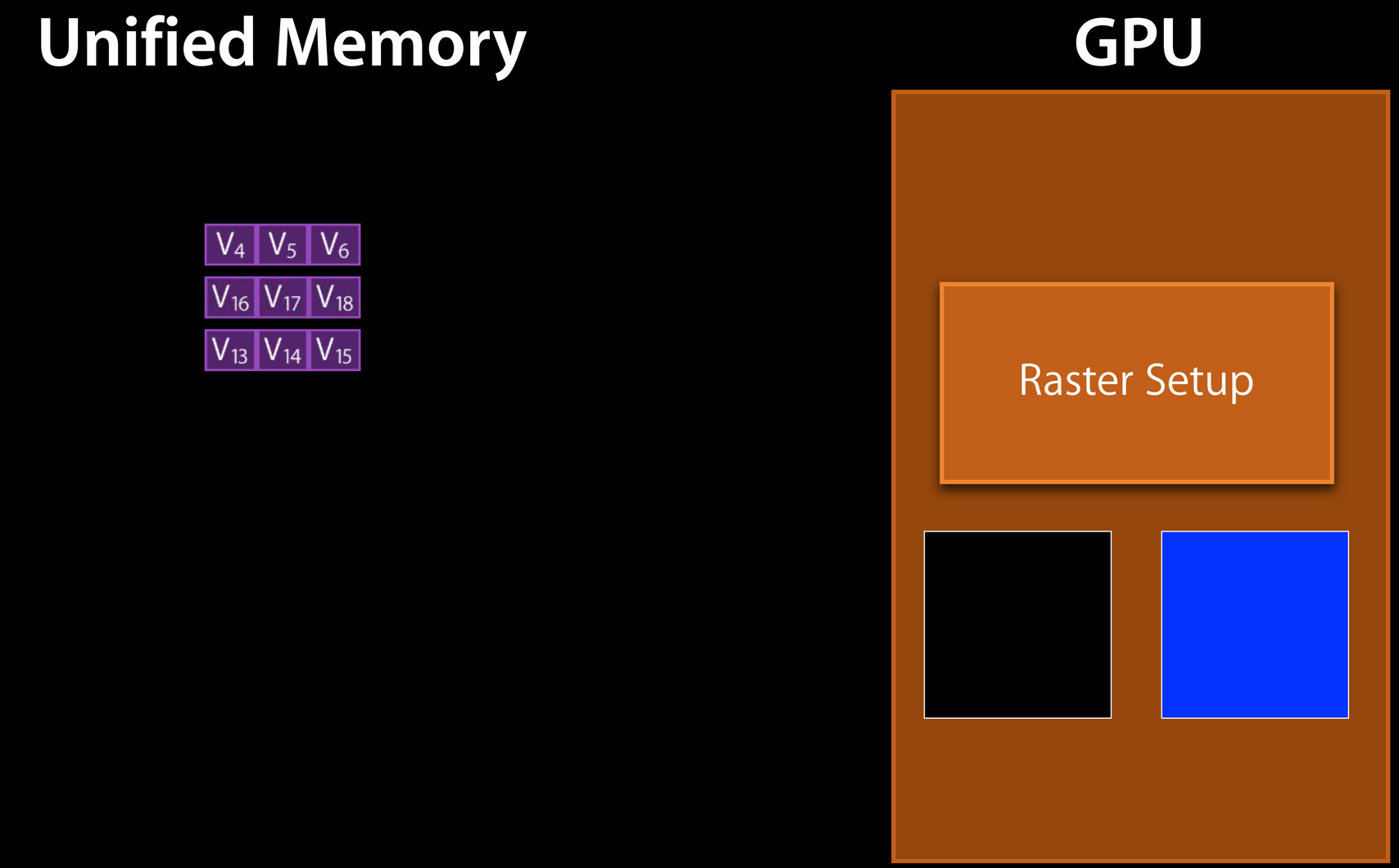
- Calling `glClear` before rendering skips Logical Buffer Load



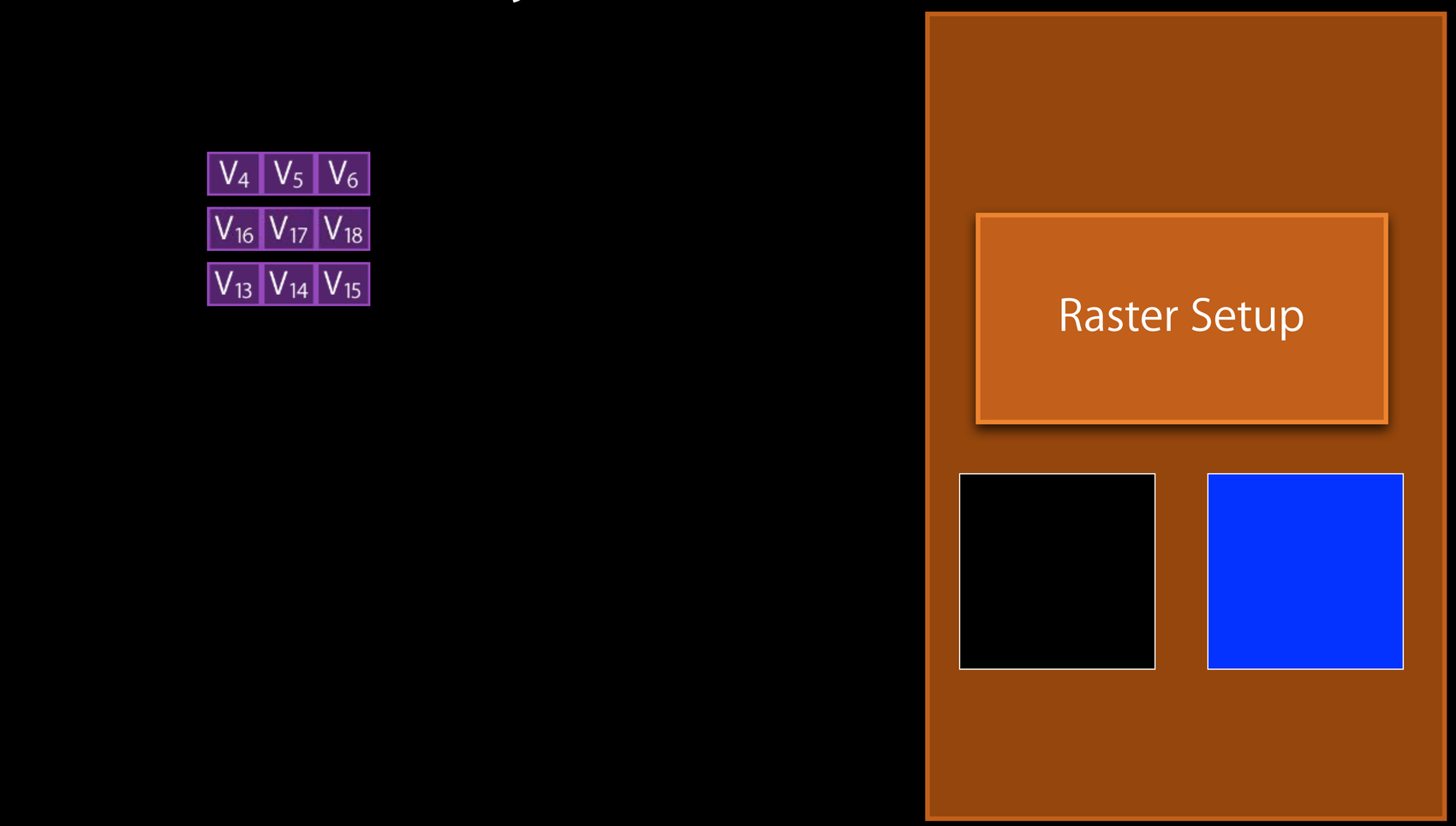
- Calling `glClear` before rendering skips Logical Buffer Load
  - Can immediately rasterize to embedded memory



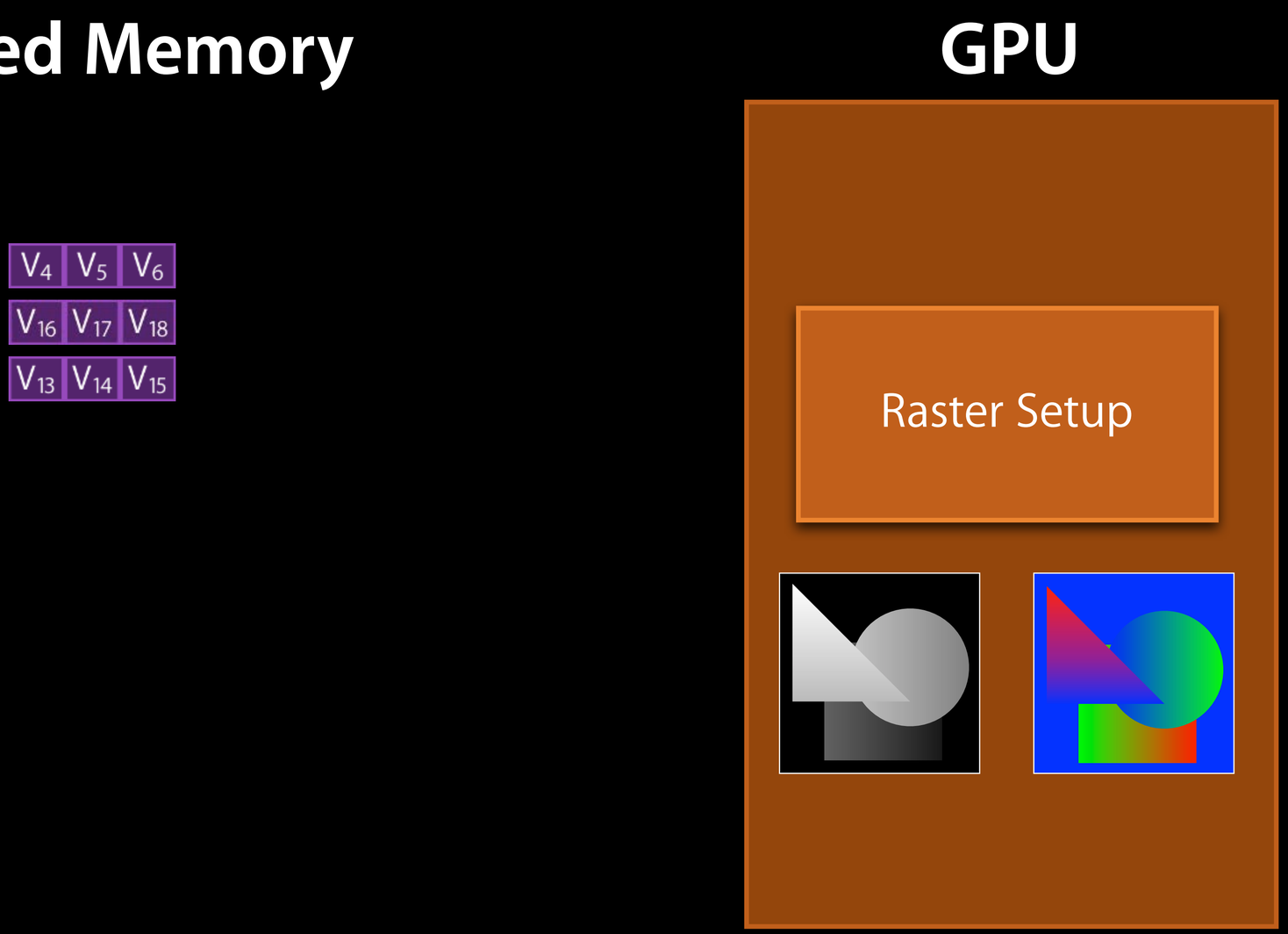
- Logical Buffer Loads also happen when switching render buffer



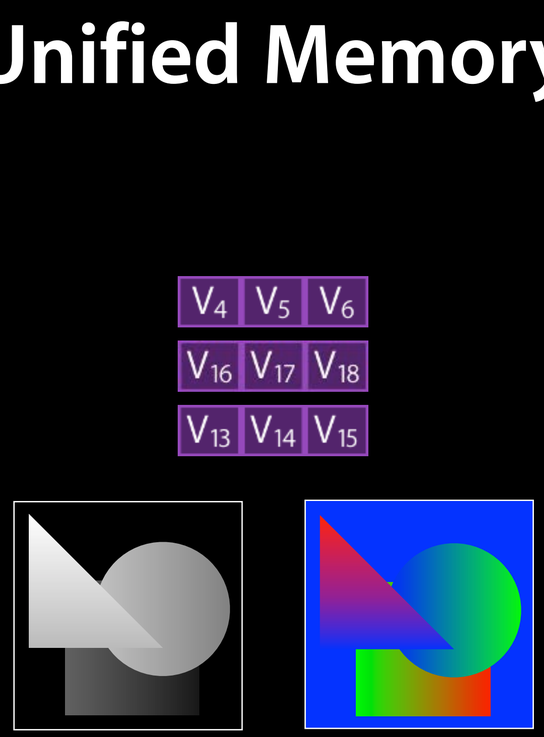
- Logical Buffer Loads also happen when switching render buffer
  - Render to texture, render to new buffer, render to texture again



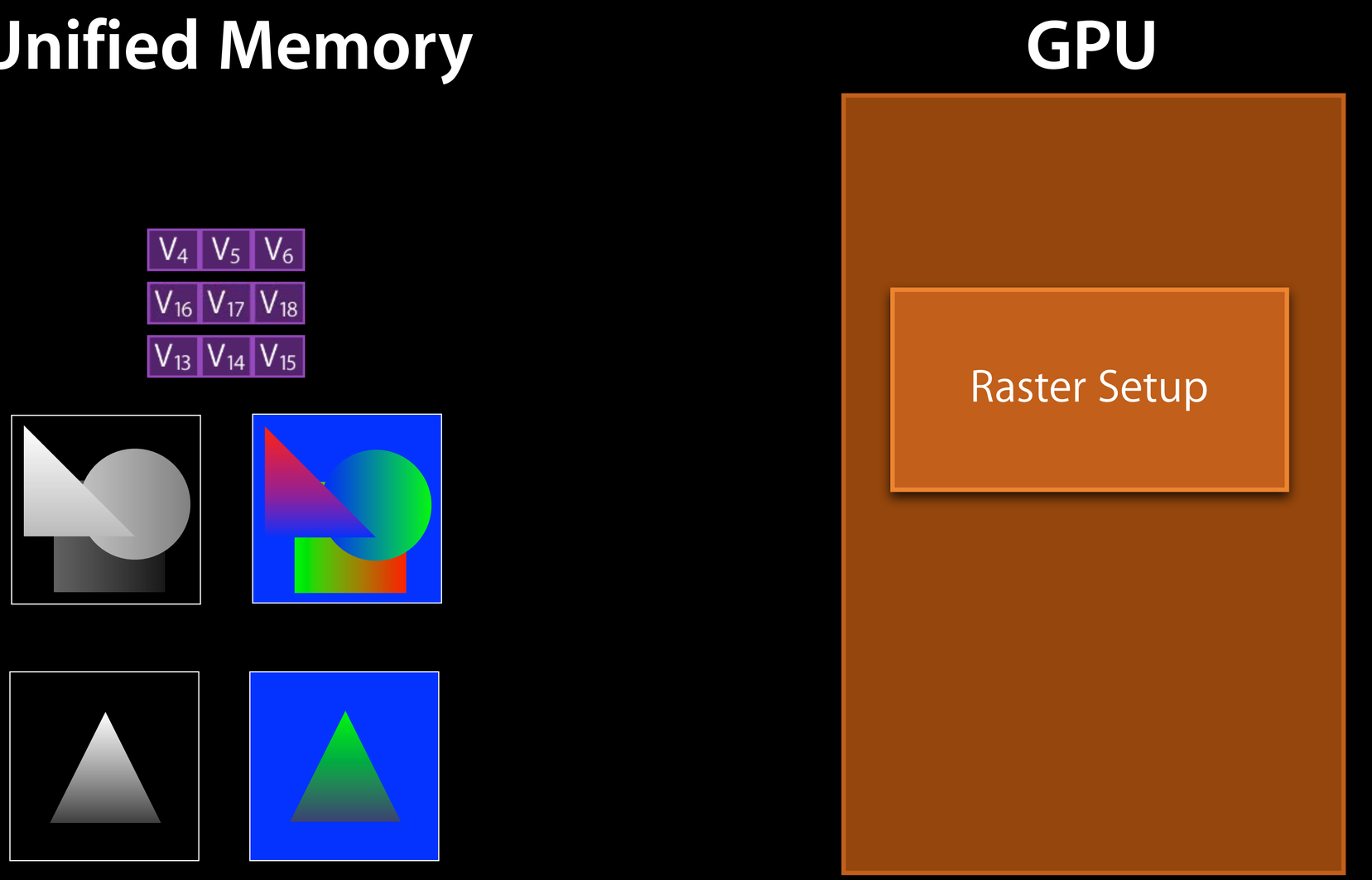
- Logical Buffer Loads also happen when switching render buffer
  - Render to texture, render to new buffer, render to texture again



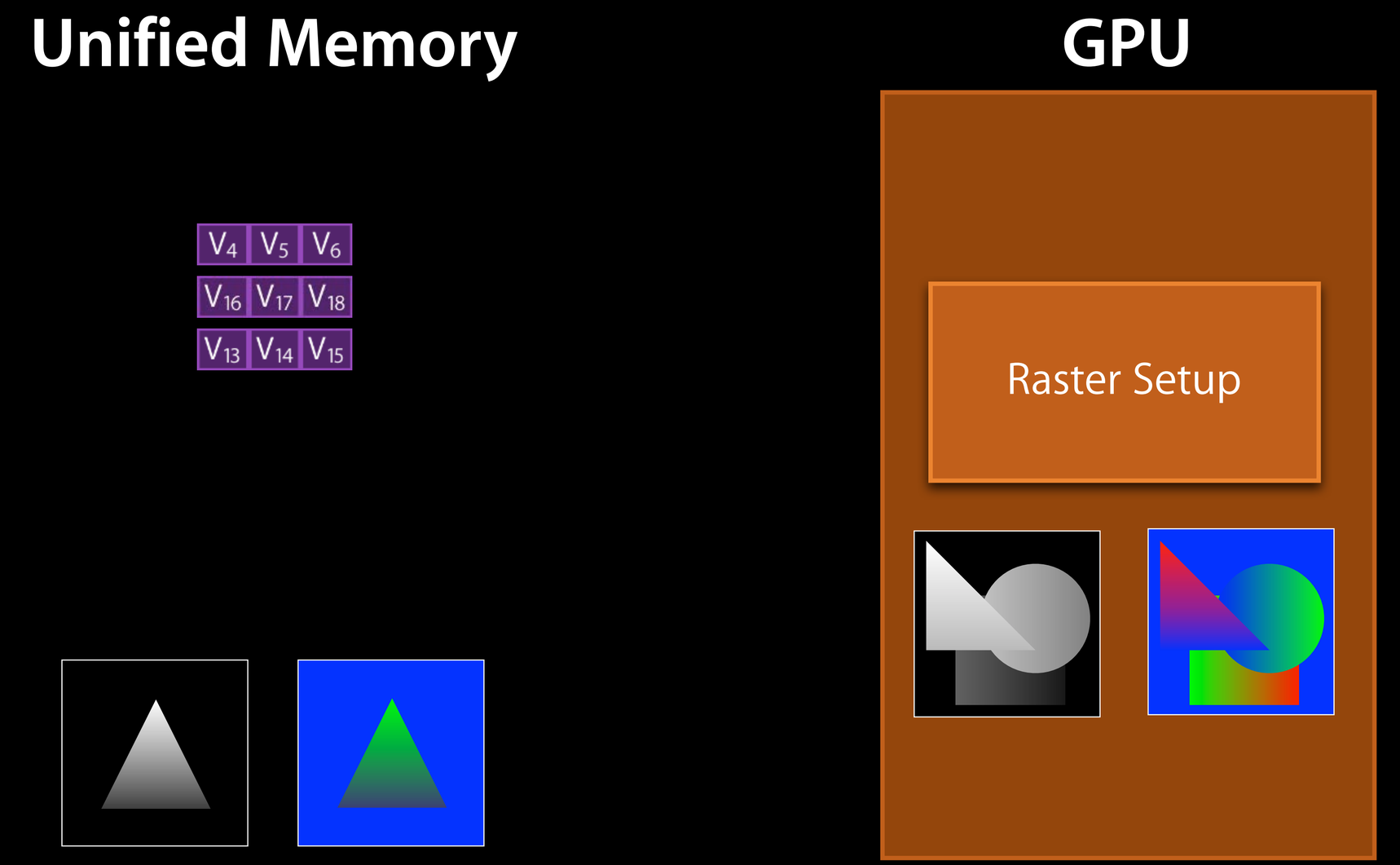
- Logical Buffer Loads also happen when switching render buffer
  - Render to texture, render to new buffer, render to texture again



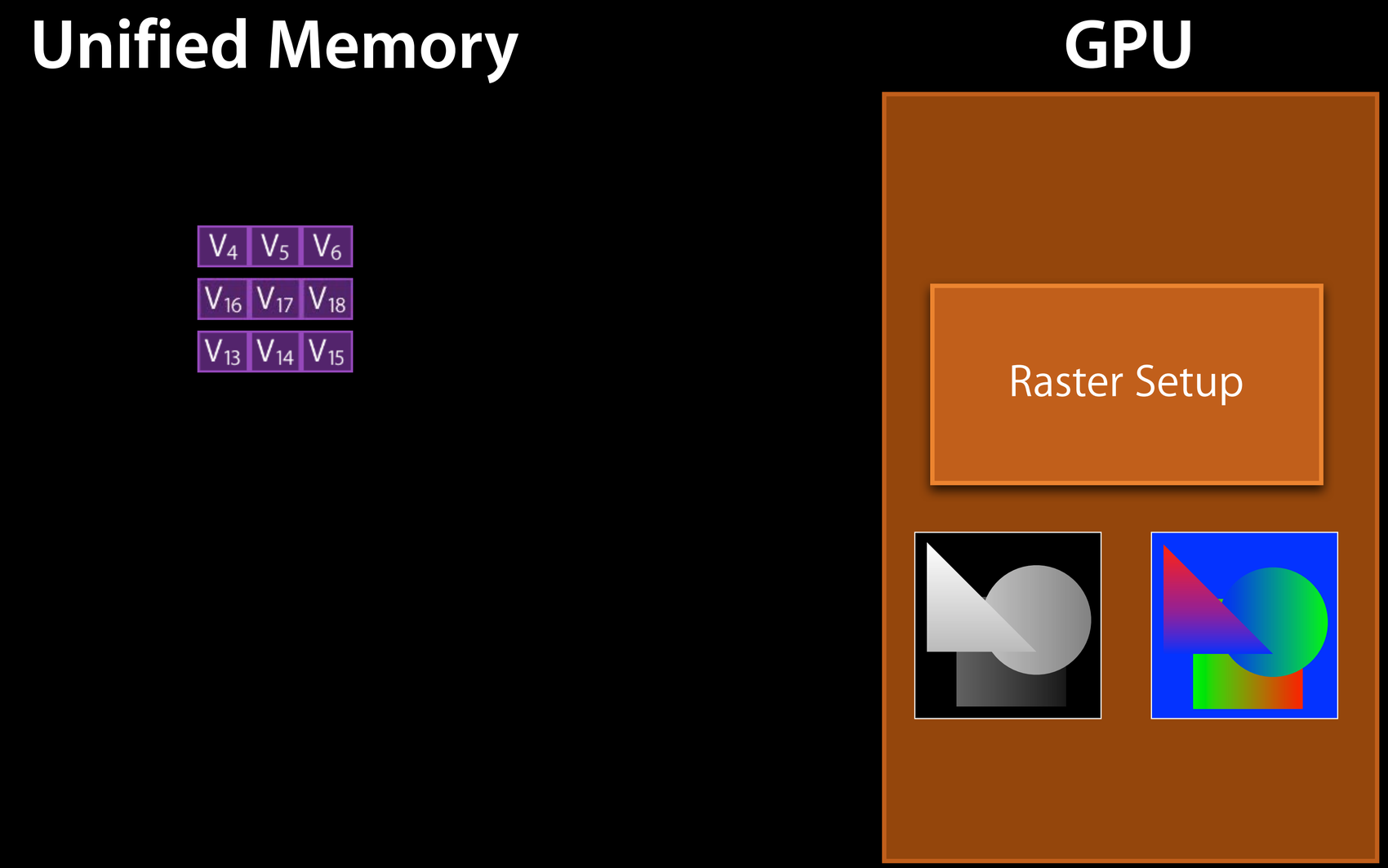
- Logical Buffer Loads also happen when switching render buffer
  - Render to texture, render to new buffer, render to texture again



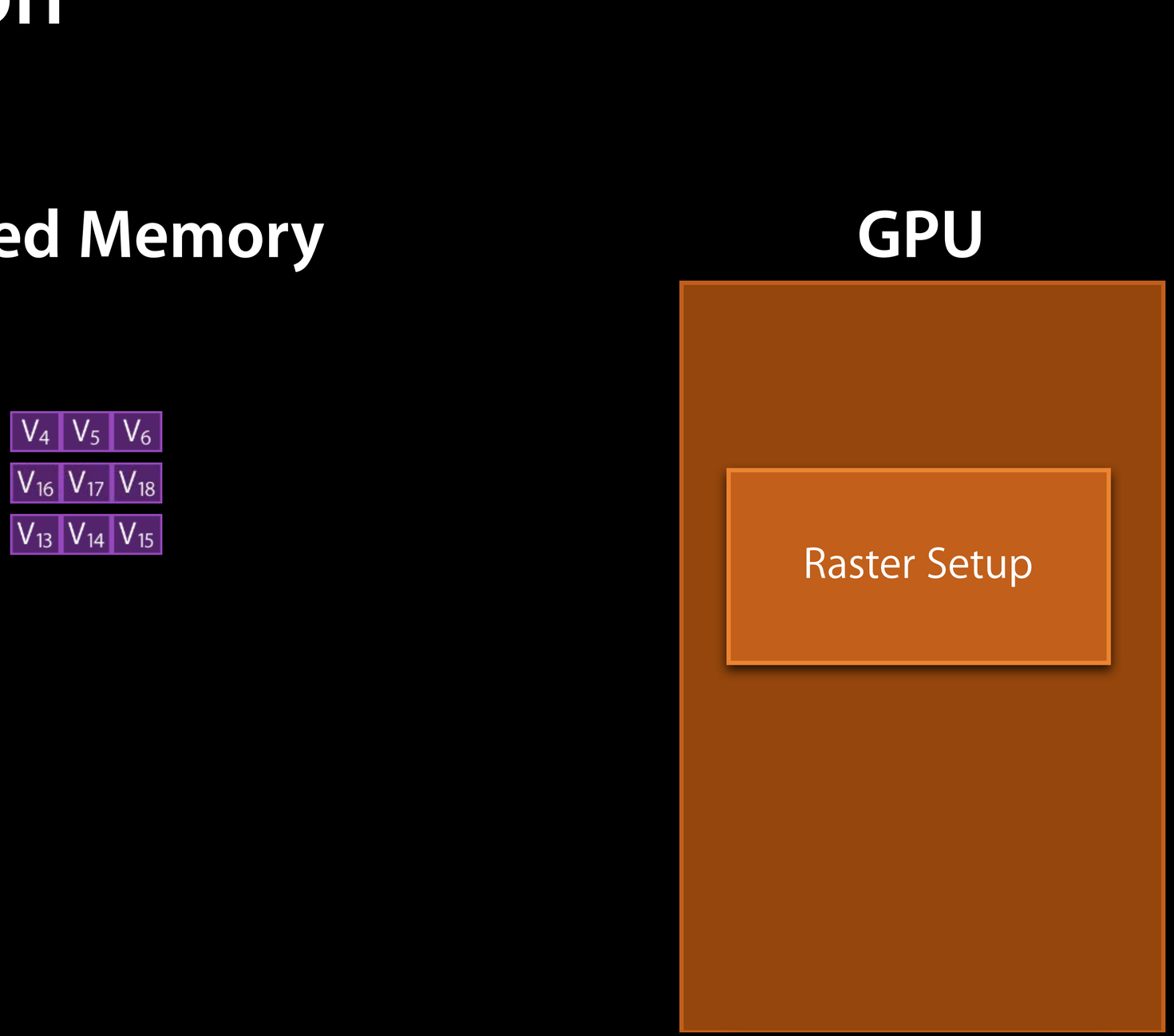
- Logical Buffer Loads also happen when switching render buffer
  - Render to texture, render to new buffer, render to texture again



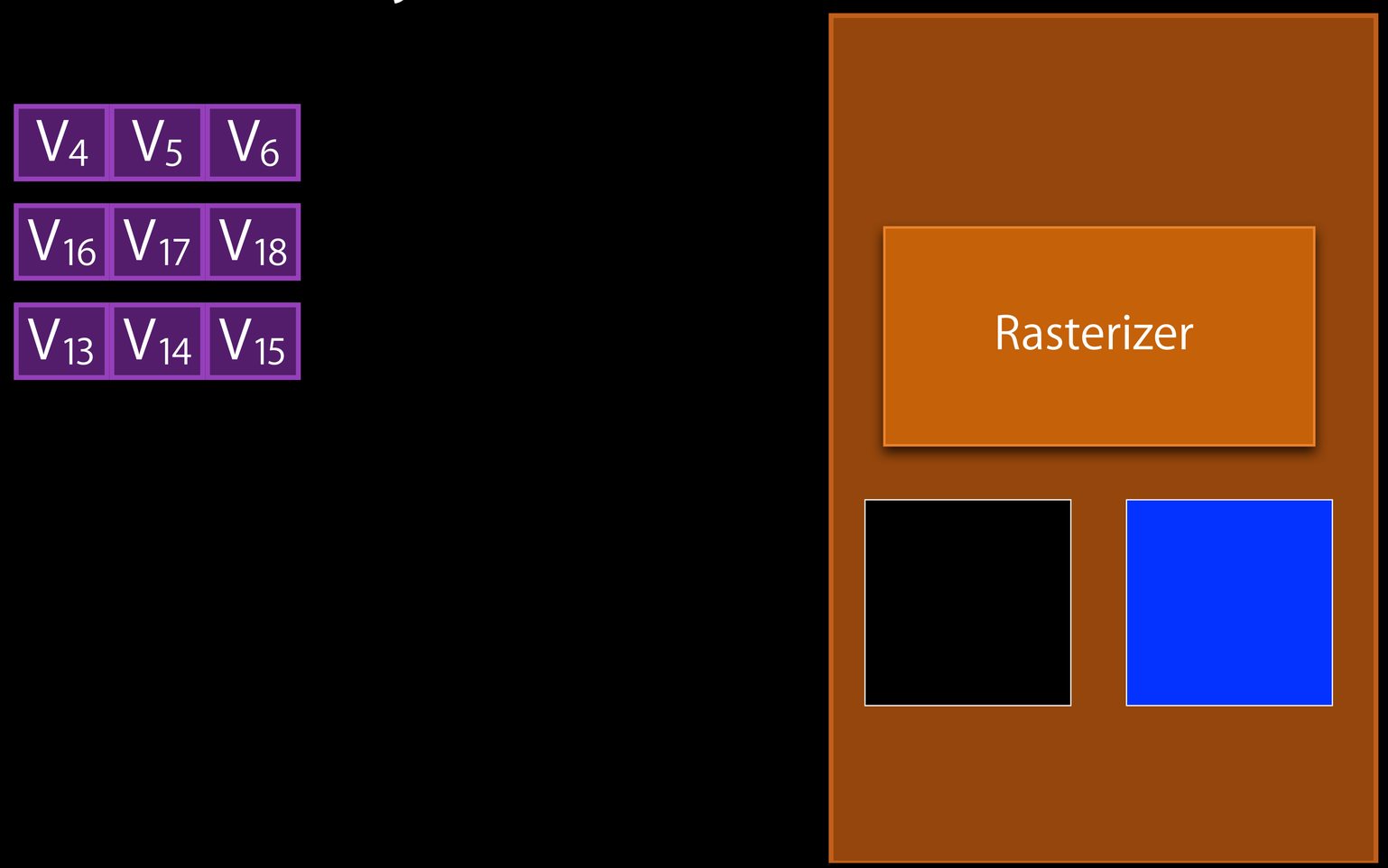
- Developers should avoid frequent switching of renderbuffers
  - Complete rendering to one buffer before switching to another



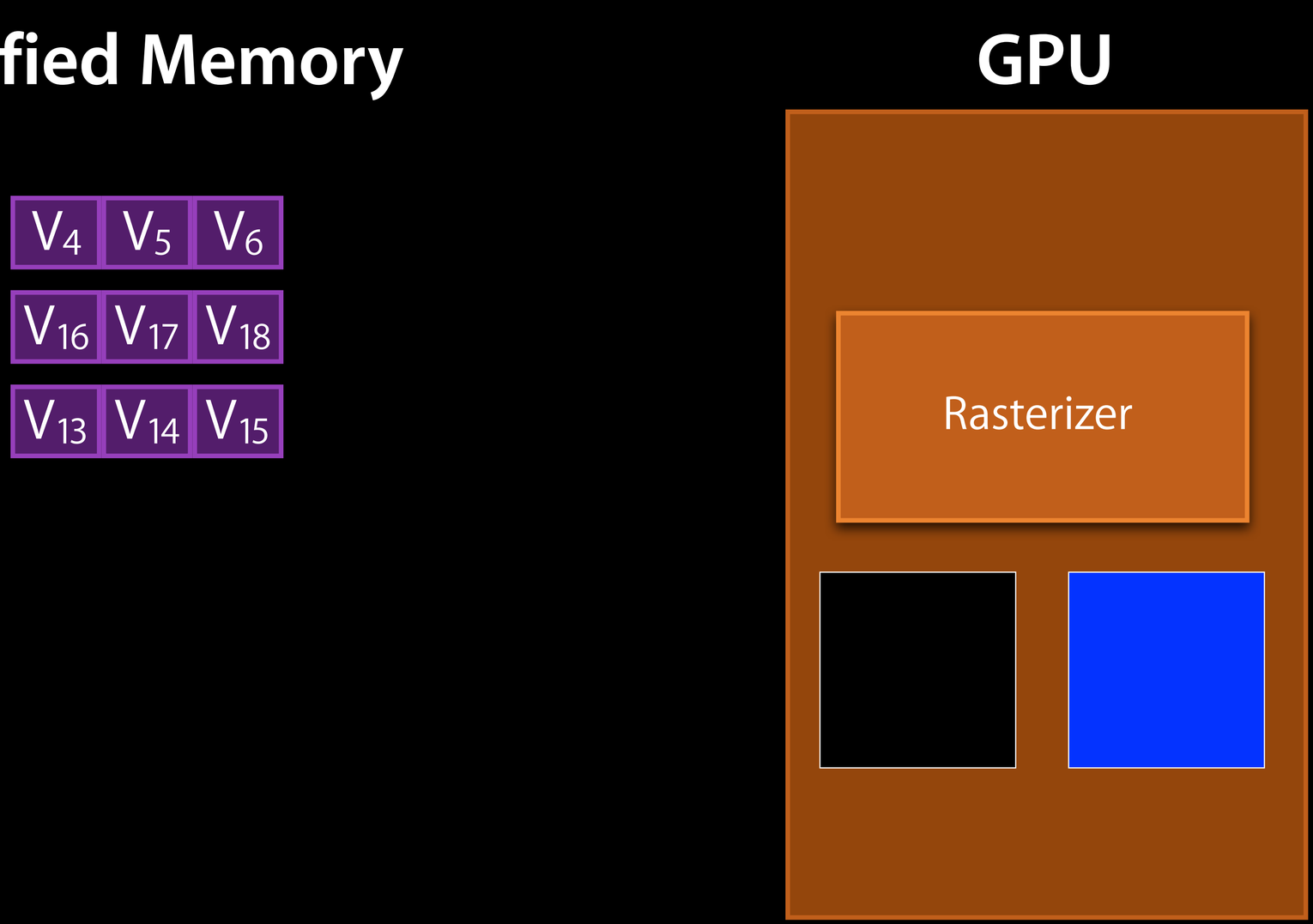
Rasterization



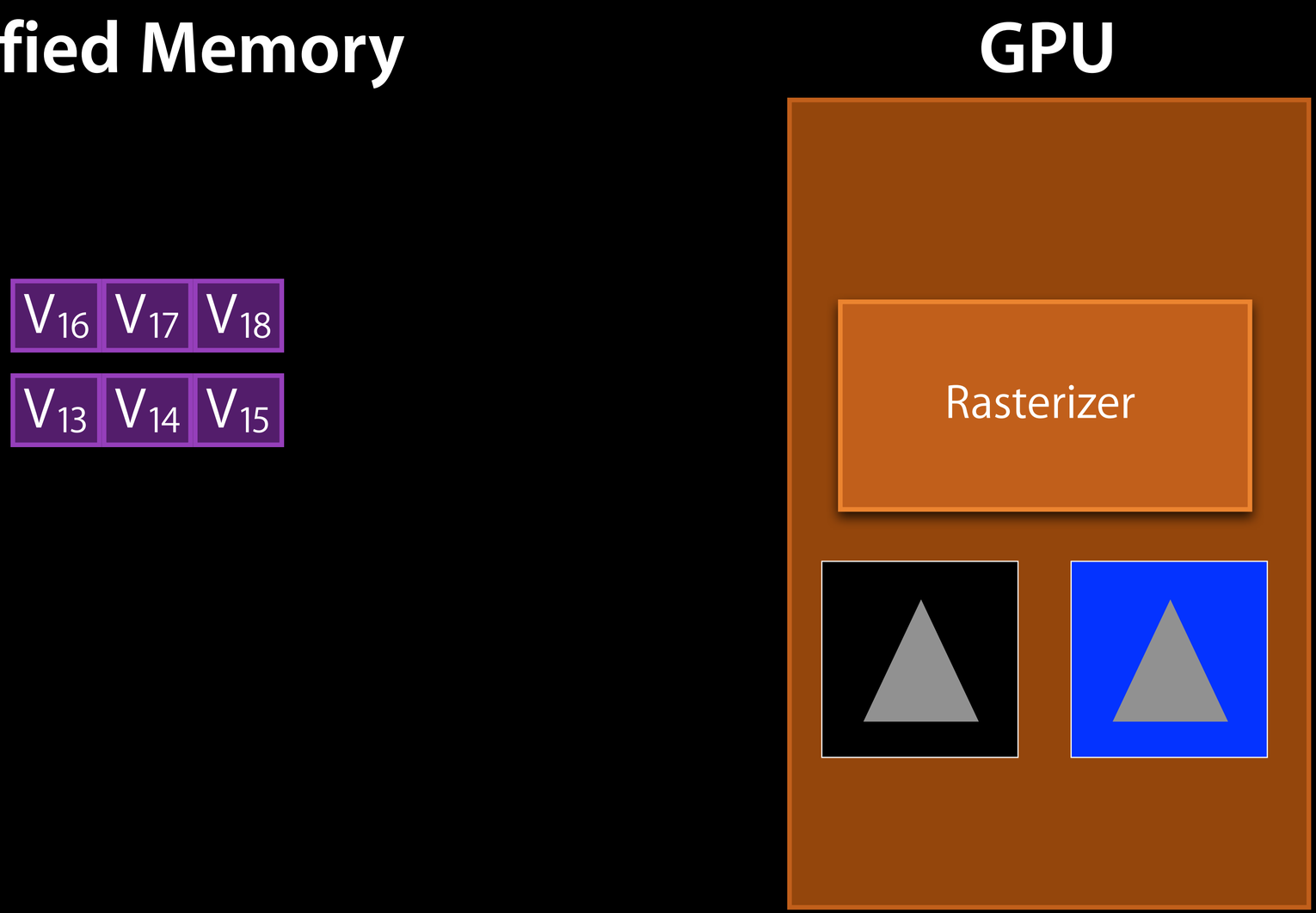
Rasterization



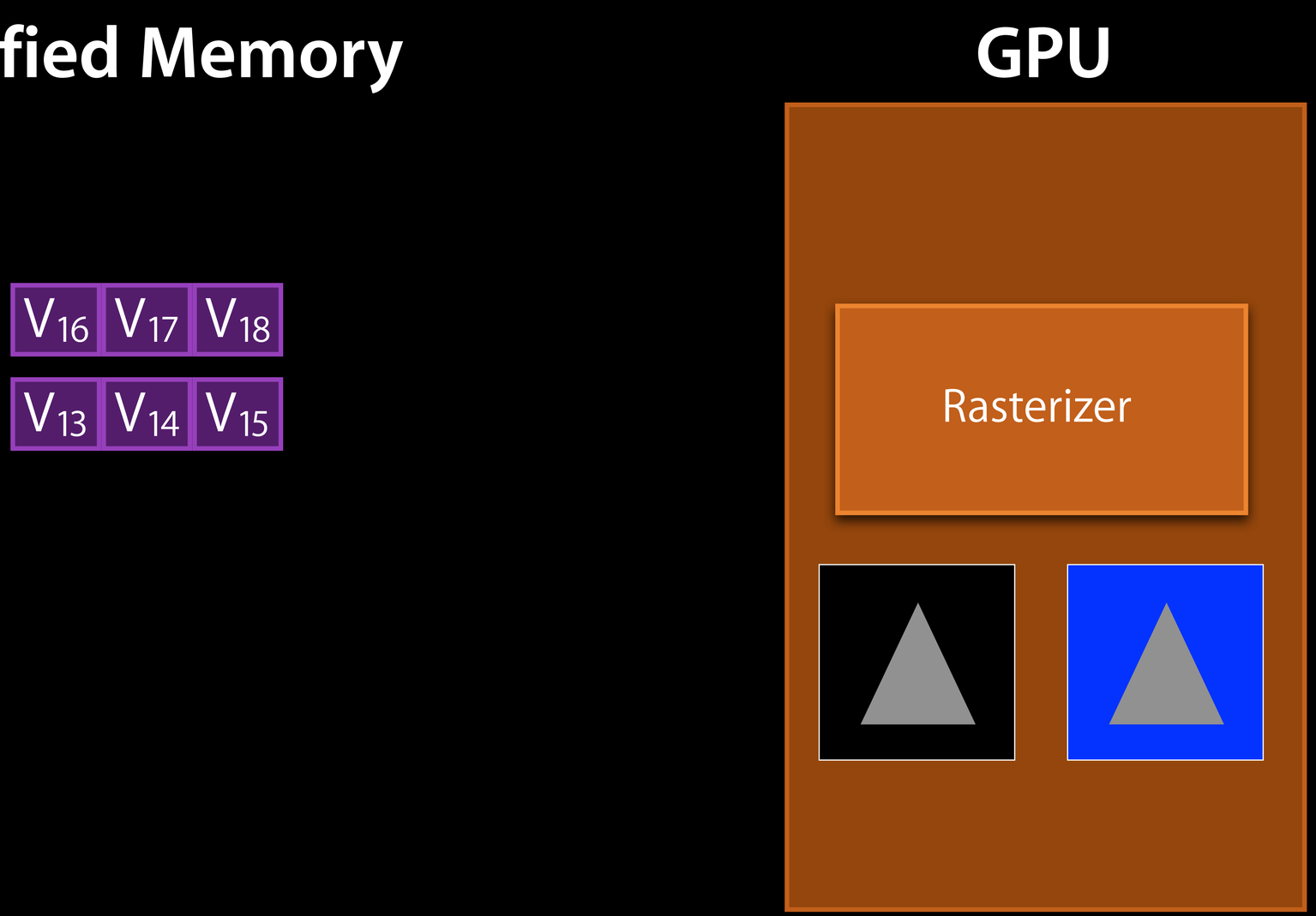
- GPU reads triangles assigned to tile
  - XY pixel coordinates and Z values generated



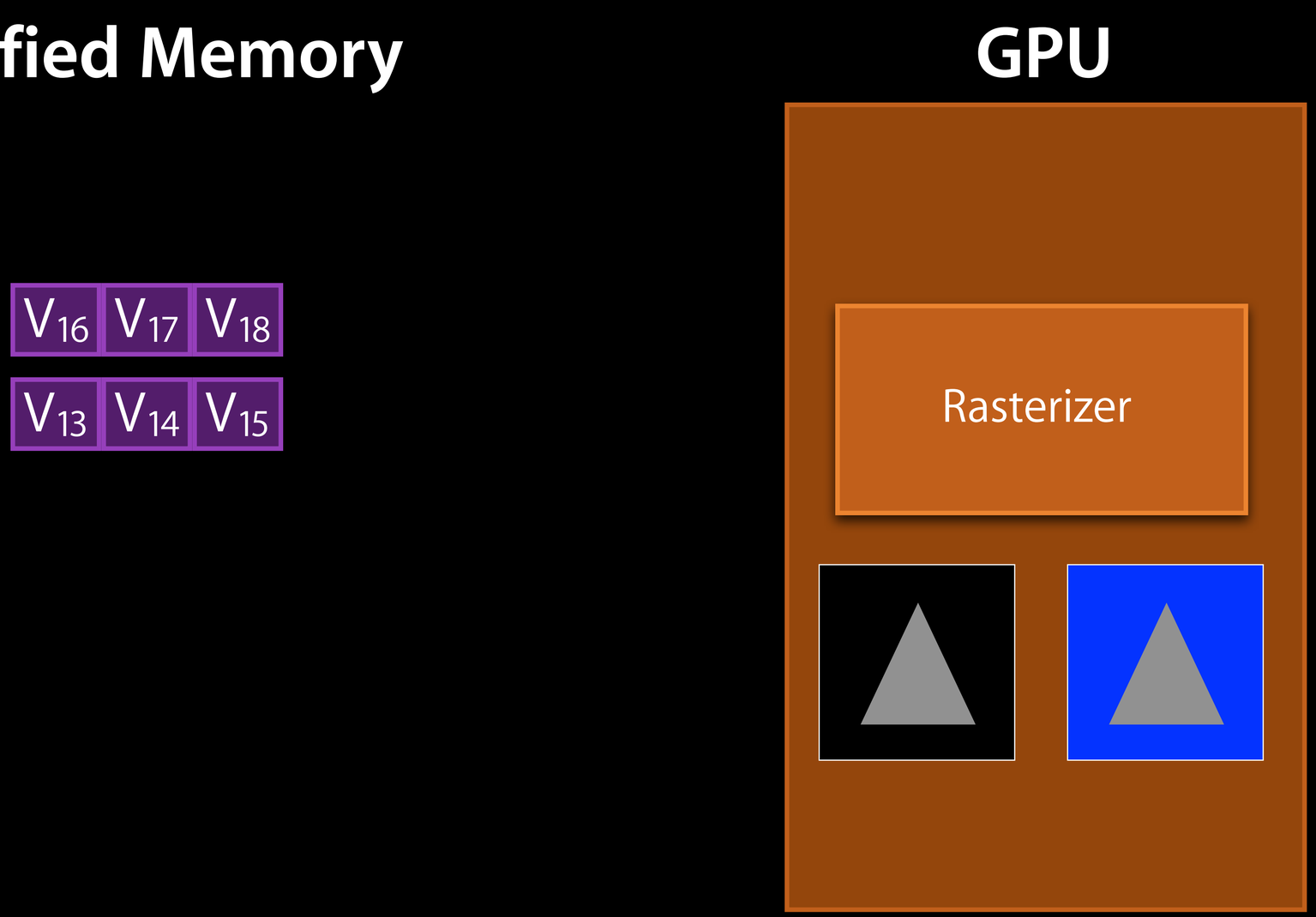
- GPU reads triangles assigned to tile
  - XY pixel coordinates and Z values generated



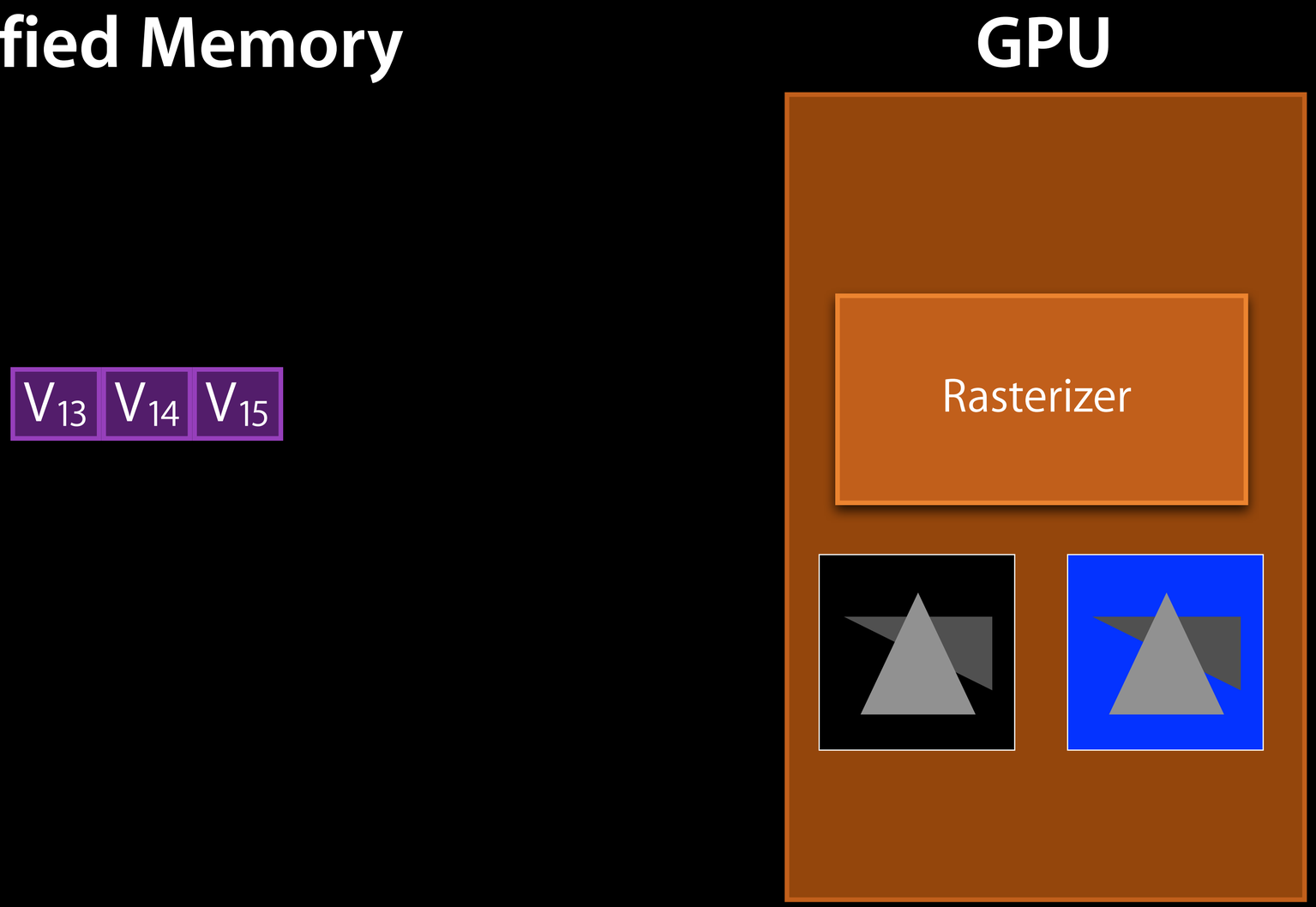
- Fragment shader not run yet
  - Positions and depth calculated only



- Hidden Surface Removal performed
  - GPU can reject fragments before shading them



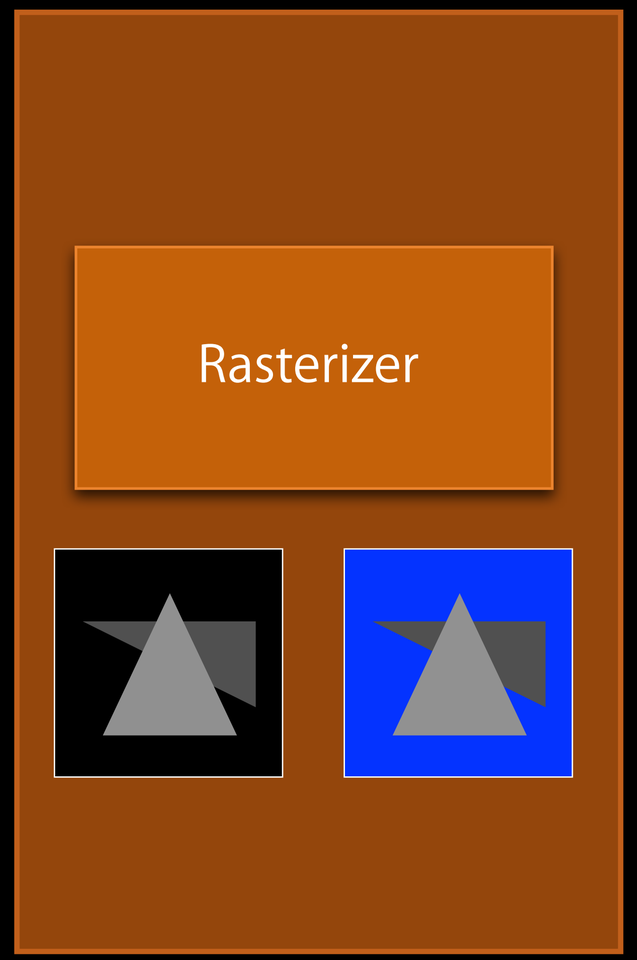
- Hidden Surface Removal performed
  - GPU can reject fragments before shading them



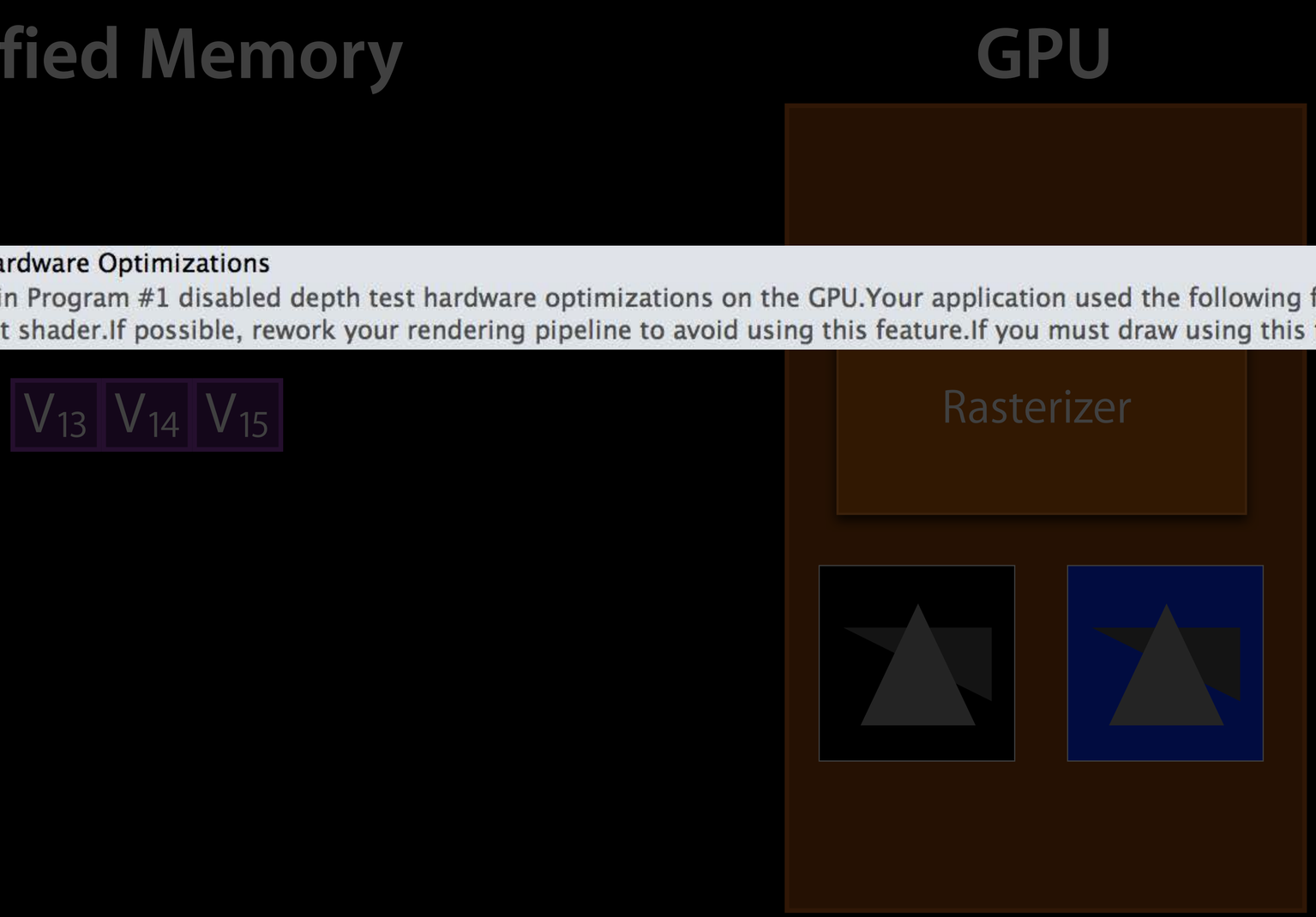
- Triangles of entire frame are present
  - Allows many fragments to be rejected

Unified Memory

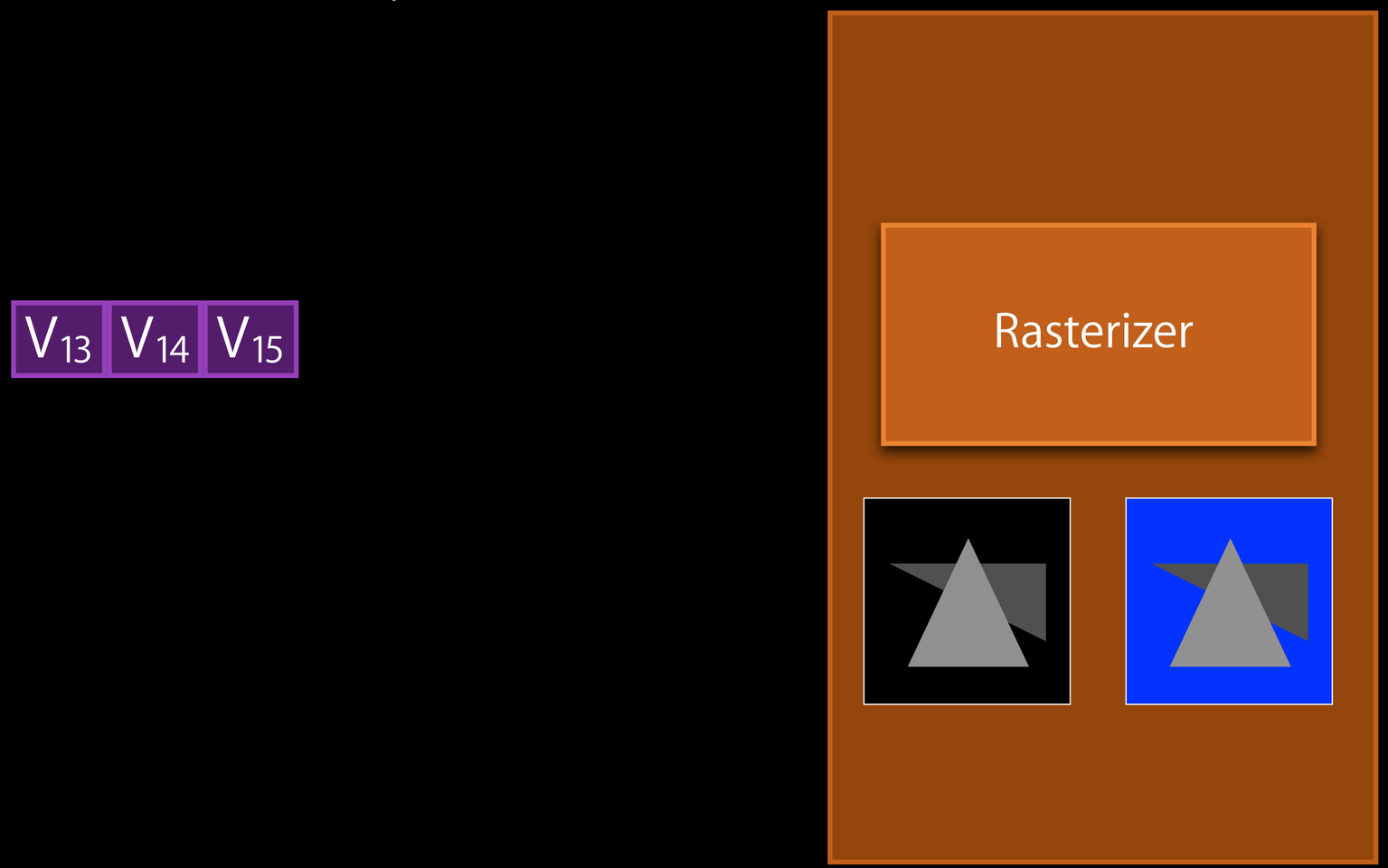
GPU



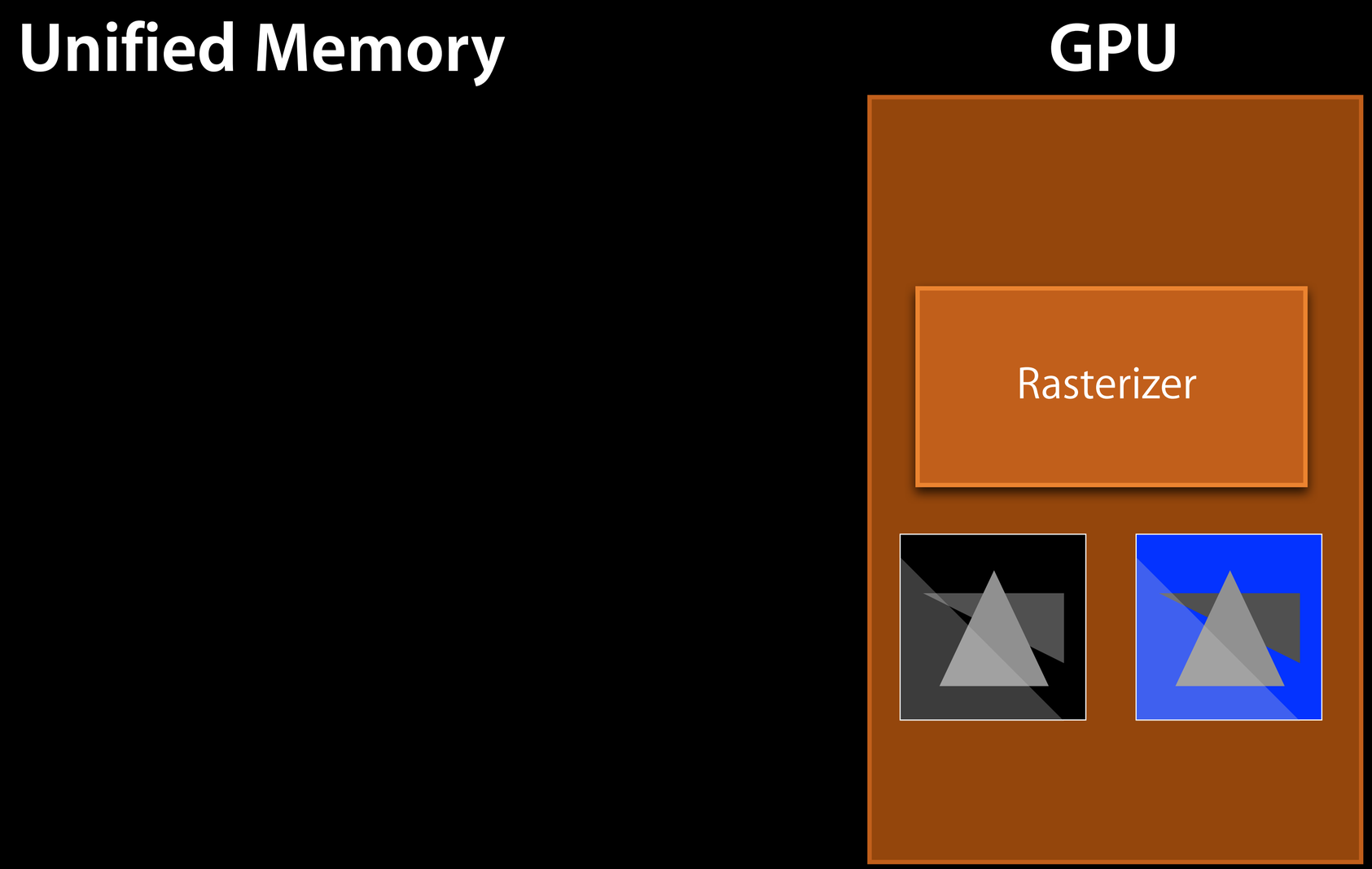
- Triangles of entire frame are present
  - Allows many fragments to be rejected



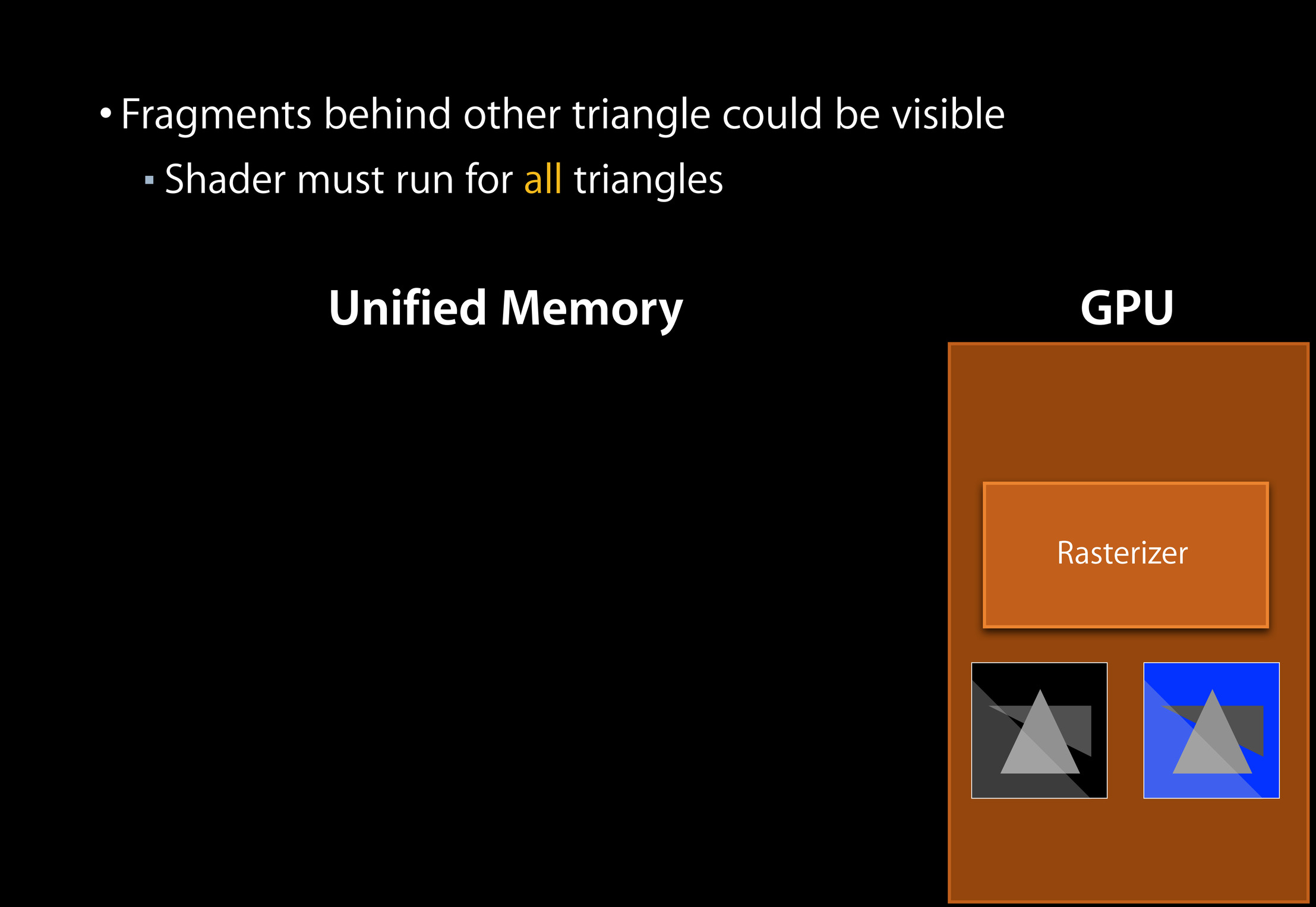
- Costly to enable blending or use `discard` in shader
  - Defeats the Hidden Surface Removal optimization



- Costly to enable blending or use `discard` in shader
  - Defeats the Hidden Surface Removal optimization



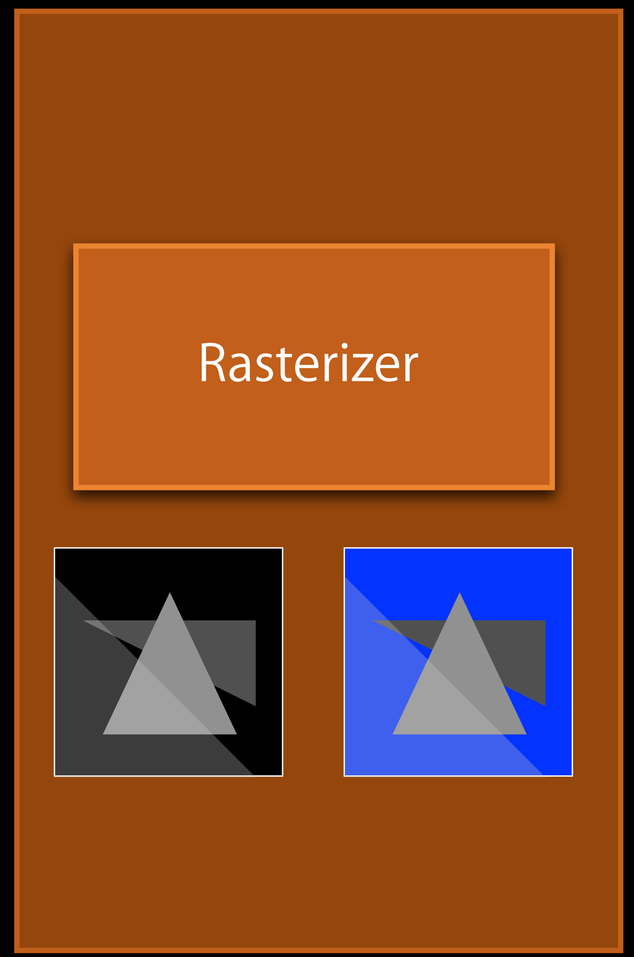
- Fragments behind other triangle could be visible
  - Shader must run for all triangles



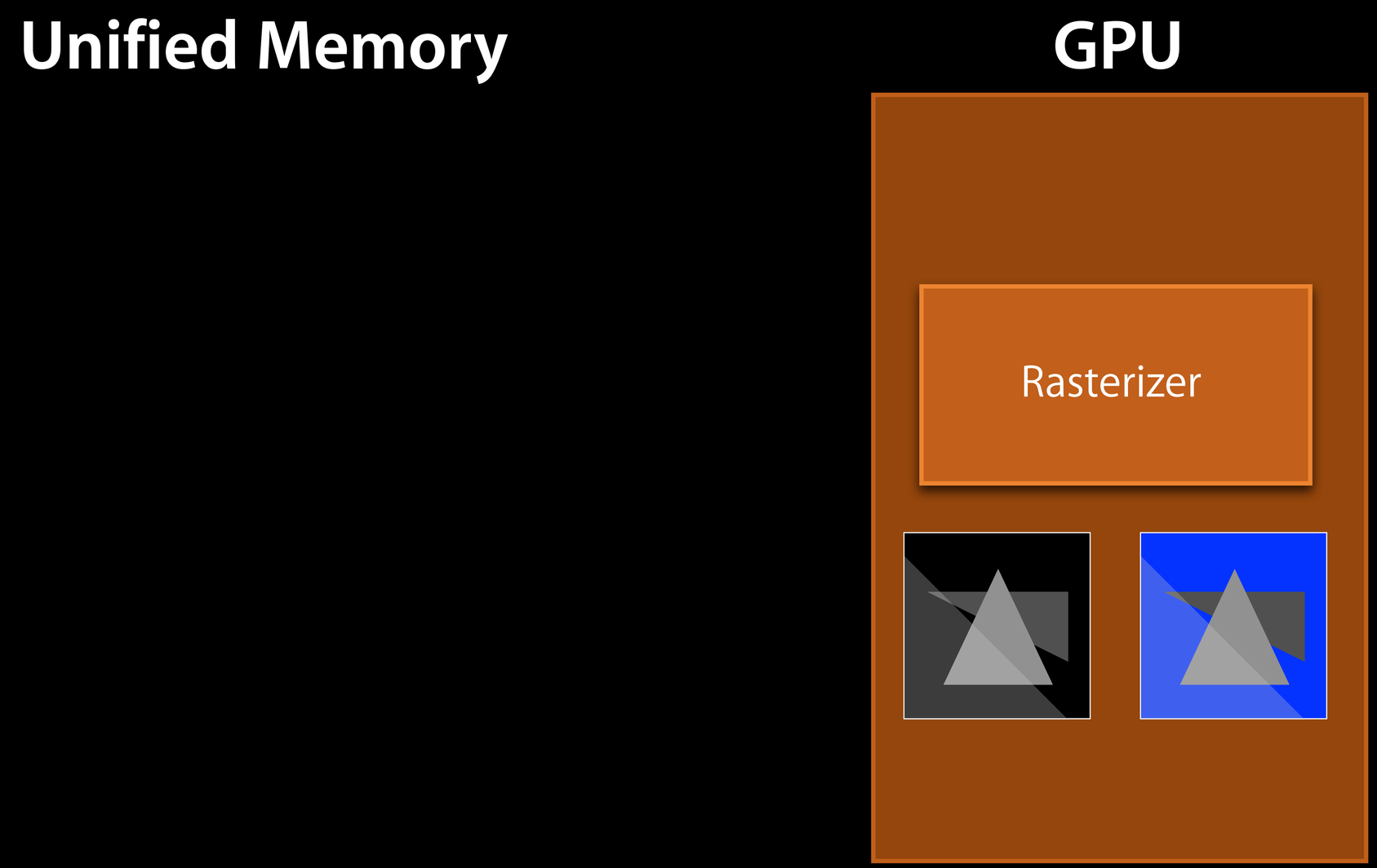
- No fragment shader savings without Hidden Surface Removal
  - Cost to performance and power

Unified Memory

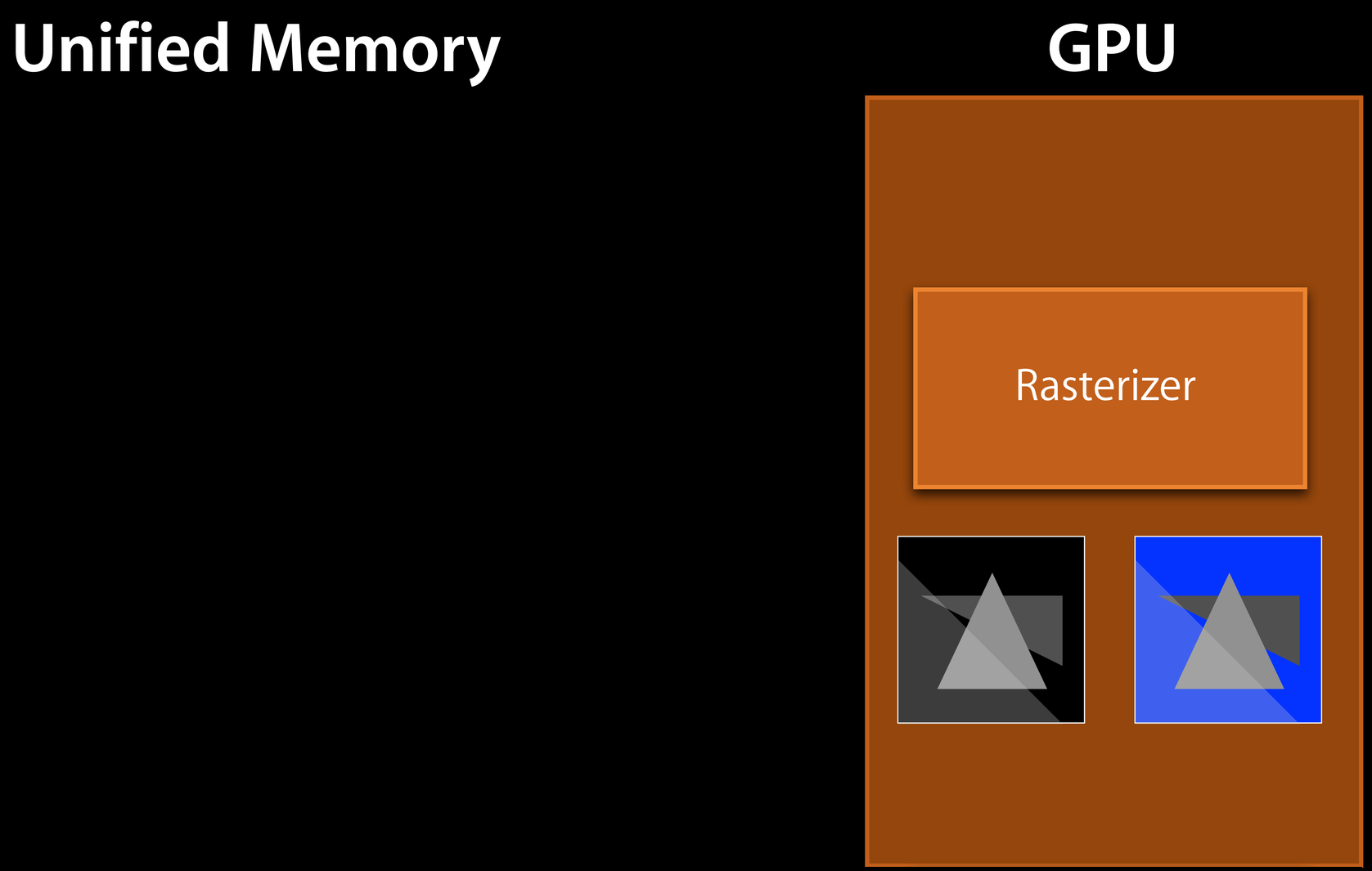
GPU



- Developers must be judicious of their use of `discard` and blending
  - Allow GPU to reject as many fragments as possible



Fragment Shading



Fragment Shading

Unified Memory

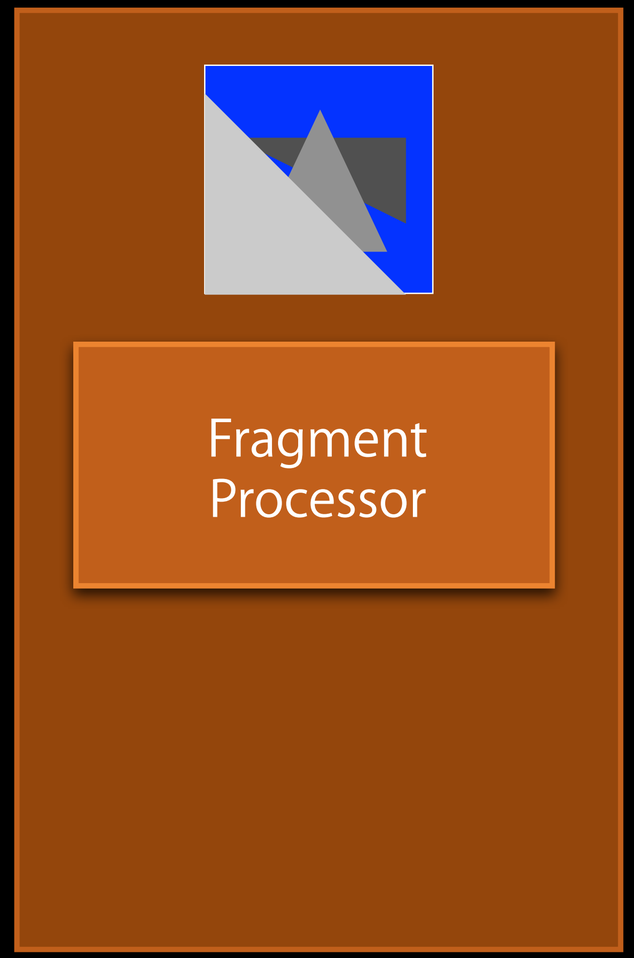
GPU

Fragment
Processor

- With Hidden Surface Removal algorithm
  - Only need to run fragment shader on each pixel once

Unified Memory

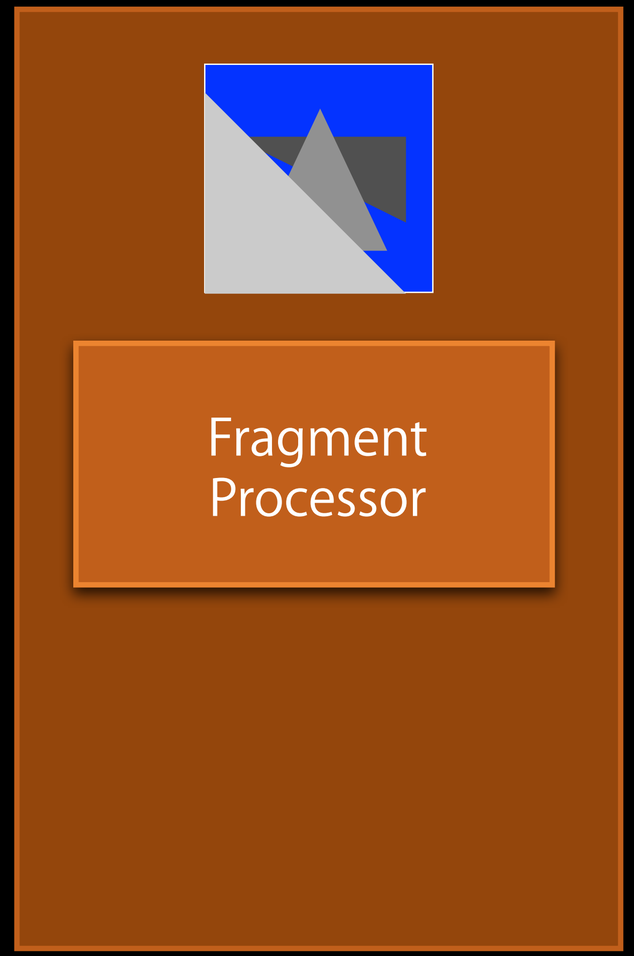
GPU



- Fragment processor shades and produces colored pixels
- Colors written to embedded tile memory on GPU

Unified Memory

GPU



- Fragment processor shades and produces colored pixels
- Colors written to embedded tile memory on GPU

Unified Memory

GPU

Fragment Processor

Tile Storage

Unified Memory

GPU

Fragment Processor

Tile Storage

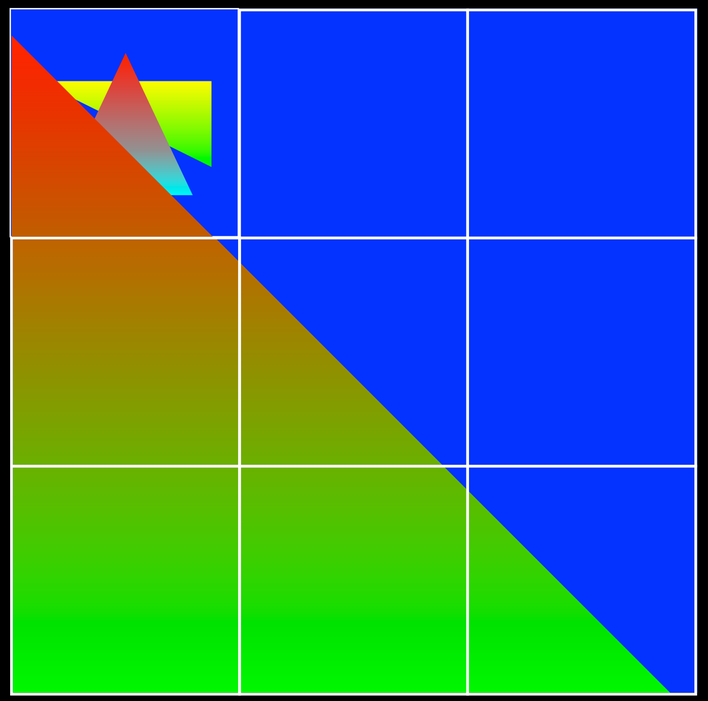
Unified Memory

GPU

Tile Store

Tile Storage

Unified Memory

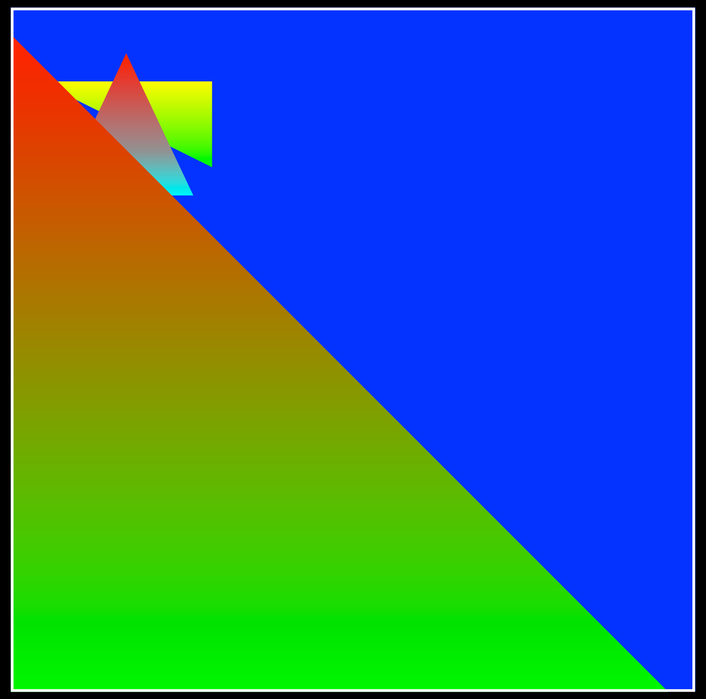


GPU

Tile Store

- Tile stored in unified memory
- Once all tiles processed, renderbuffer is ready for use

Unified Memory

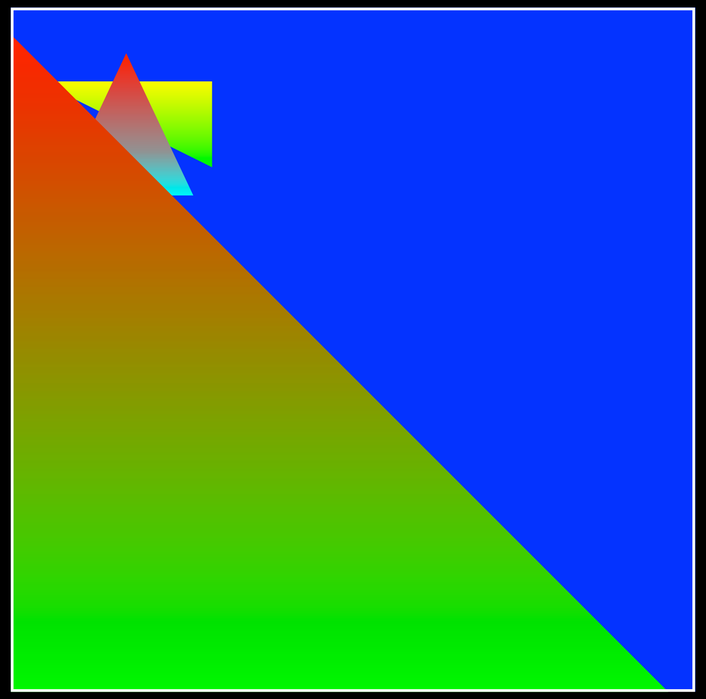


GPU

Tile Store

- Storing tile to unified memory called Logical Buffer Store
- Each frame needs at least one

Unified Memory

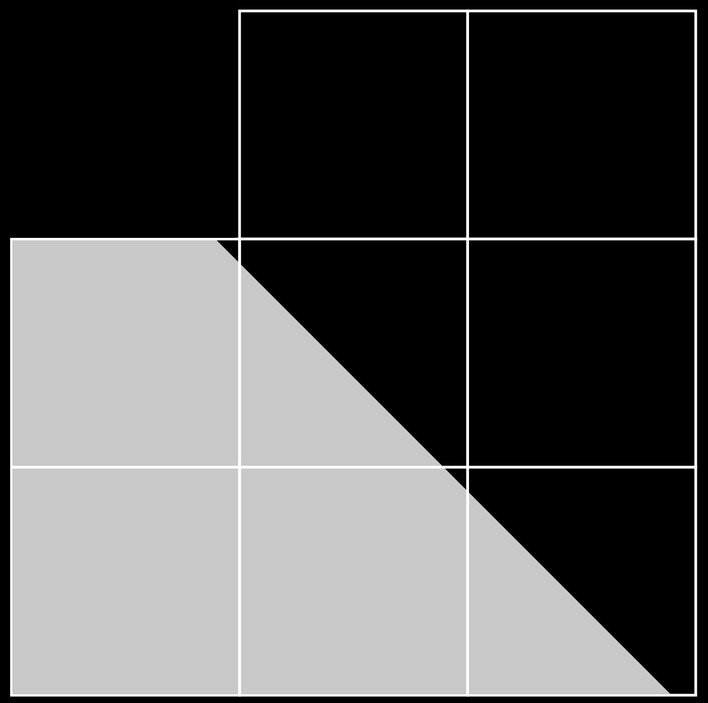


GPU

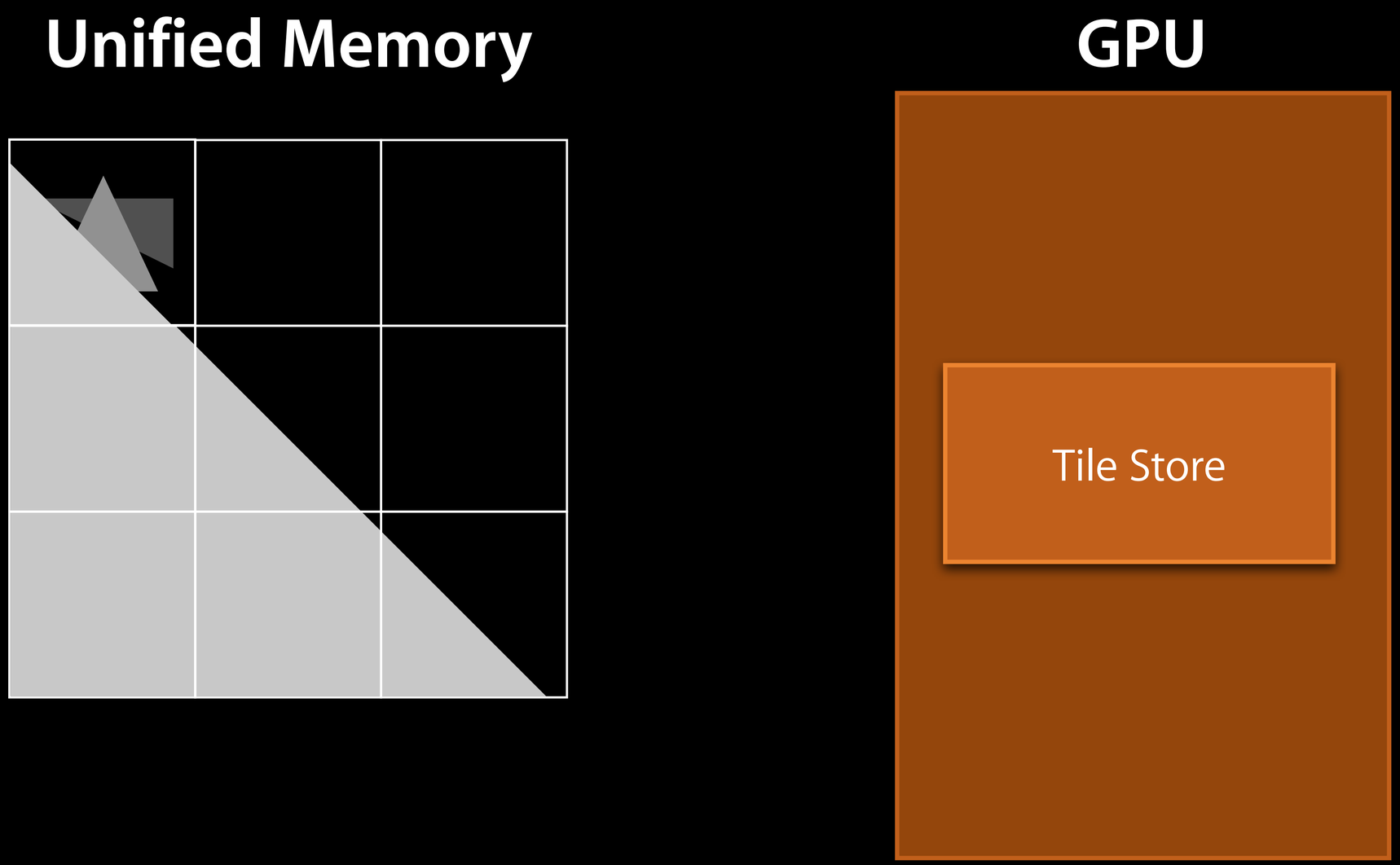
Tile Store

- Storing tile to unified memory called Logical Buffer Store
- Each frame needs at least one

- Depth only needs to be stored for depth texture effects
  - Such as shadowing or Screen Space Ambient Occlusion

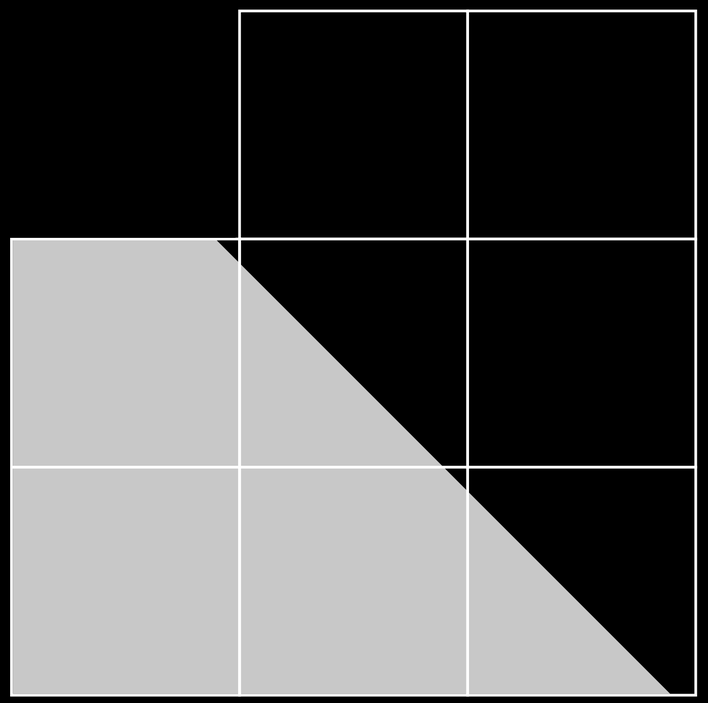


- Depth only needs to be stored for depth texture effects
  - Such as shadowing or Screen Space Ambient Occlusion



- Unless using such effect
  - Depth buffer storage unnecessary

Unified Memory



GPU

Tile Store

- Unless using such effect
  - Depth buffer storage unnecessary

Unified Memory

GPU

Tile Store

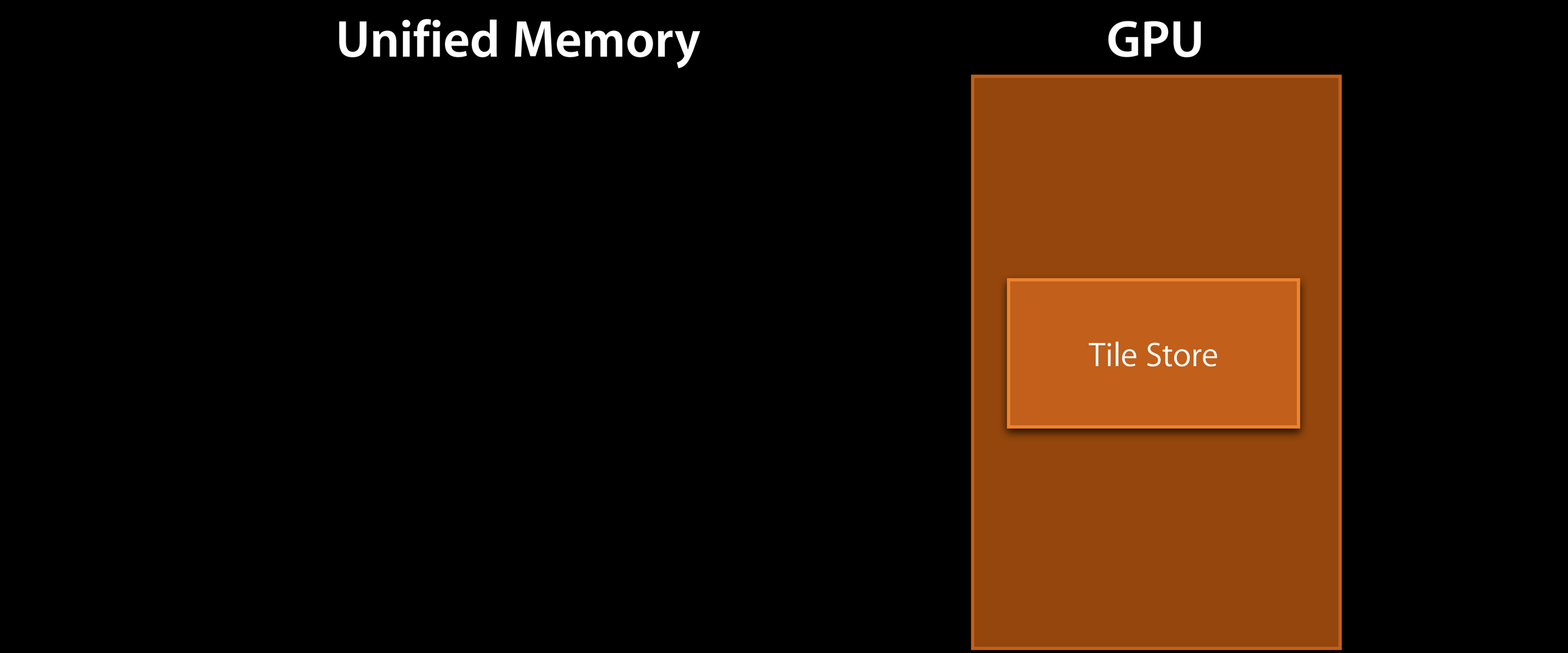
- Developers can call `glDiscardFramebufferEXT` to skip Logical Buffer Store
- Depth buffer tile discarded after rendering complete

Unified Memory

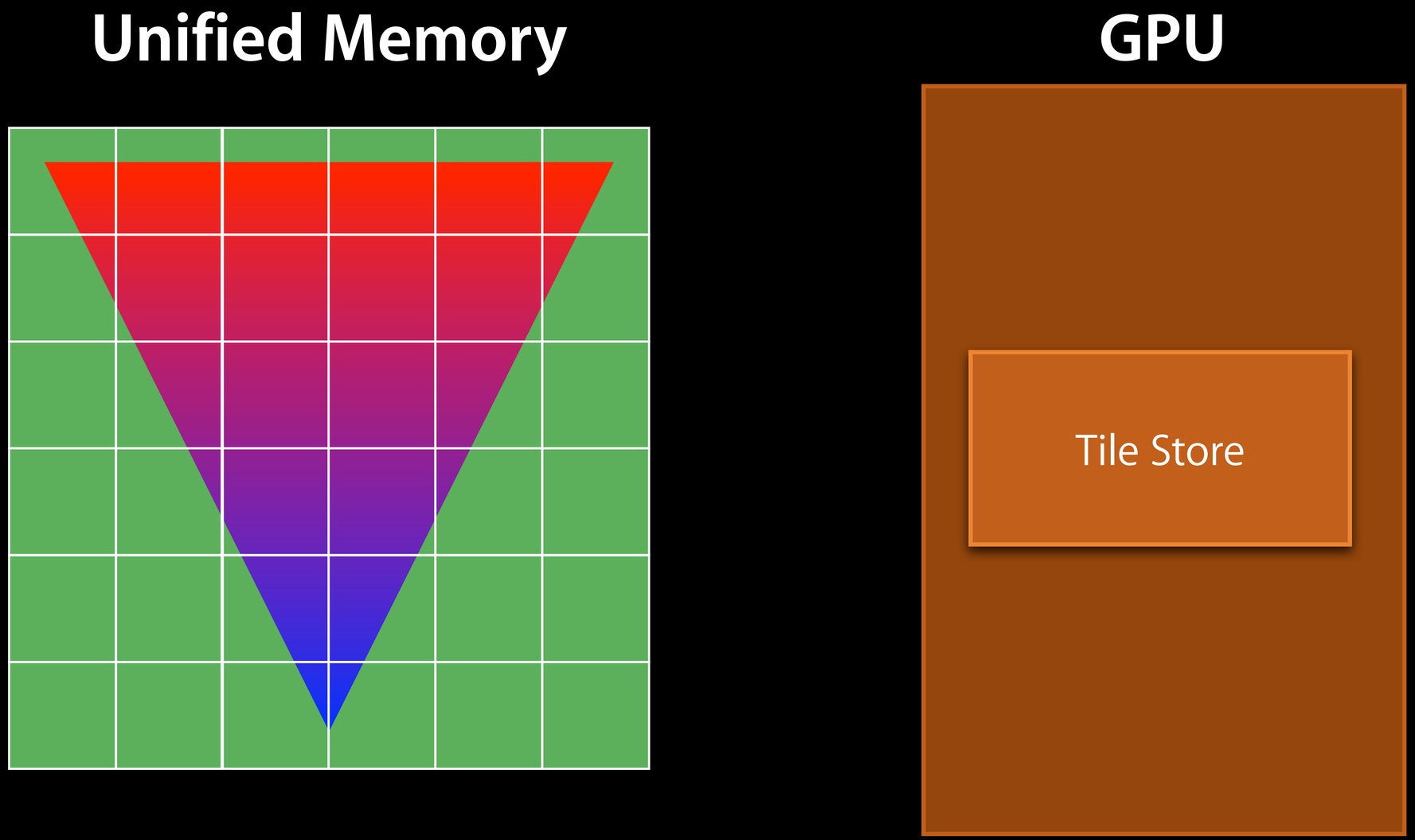
GPU

Tile Store

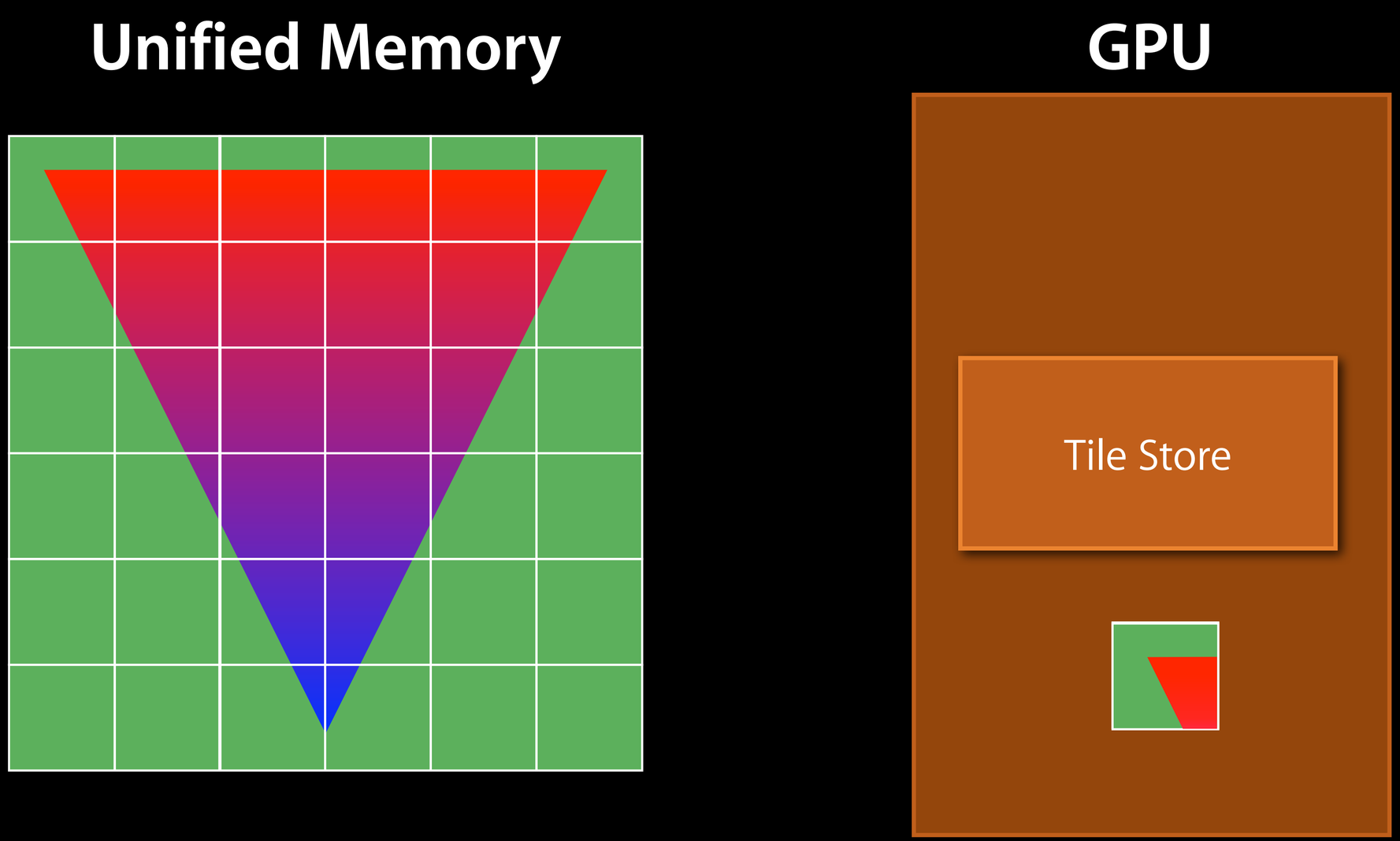
- Developers can call `glDiscardFramebufferEXT` to skip Logical Buffer Store
- Depth buffer tile discarded after rendering complete



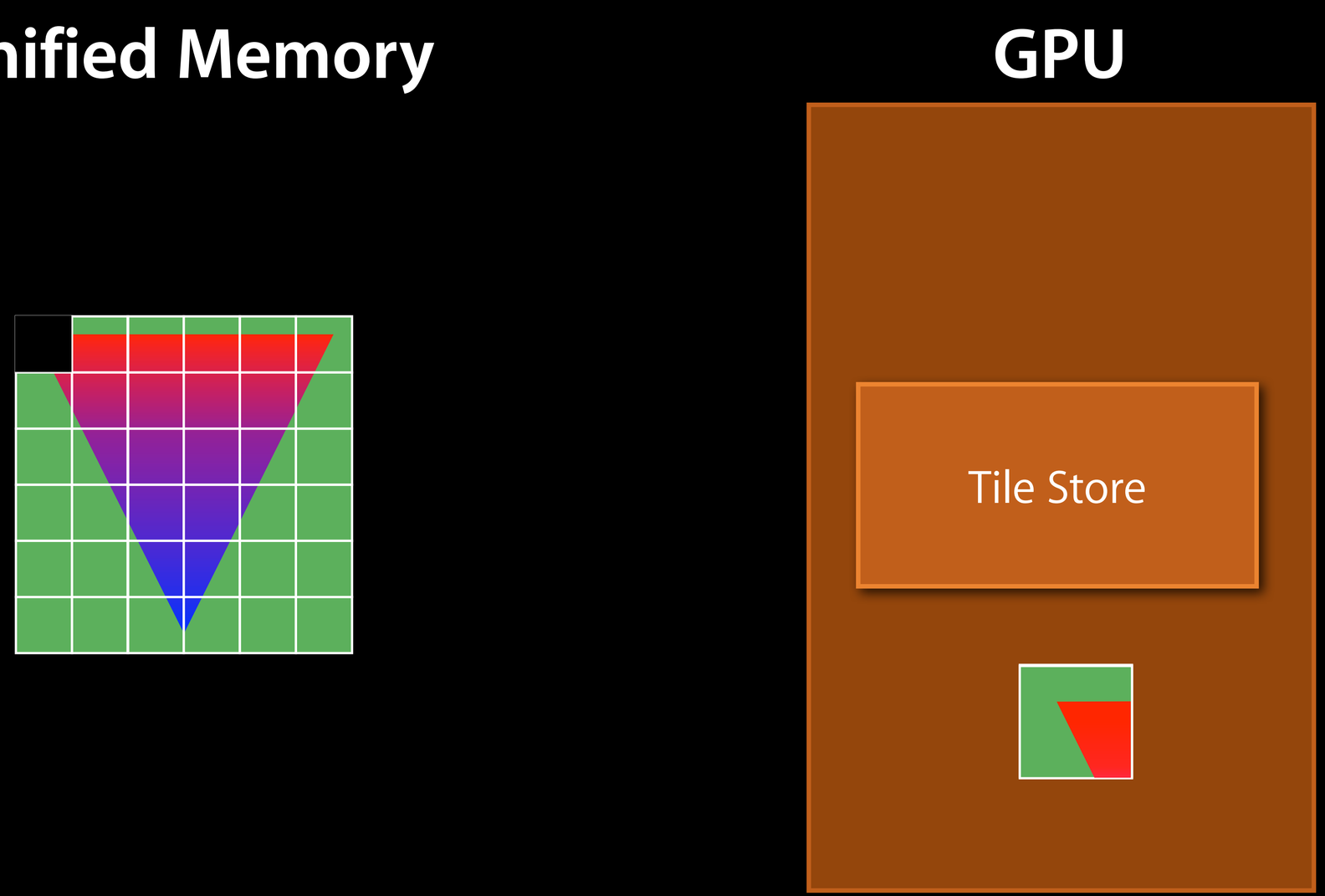
- Multisample anti-aliased renderbuffers have many more tiles
  - Developers do not need to store preresolved MSAA renderbuffer



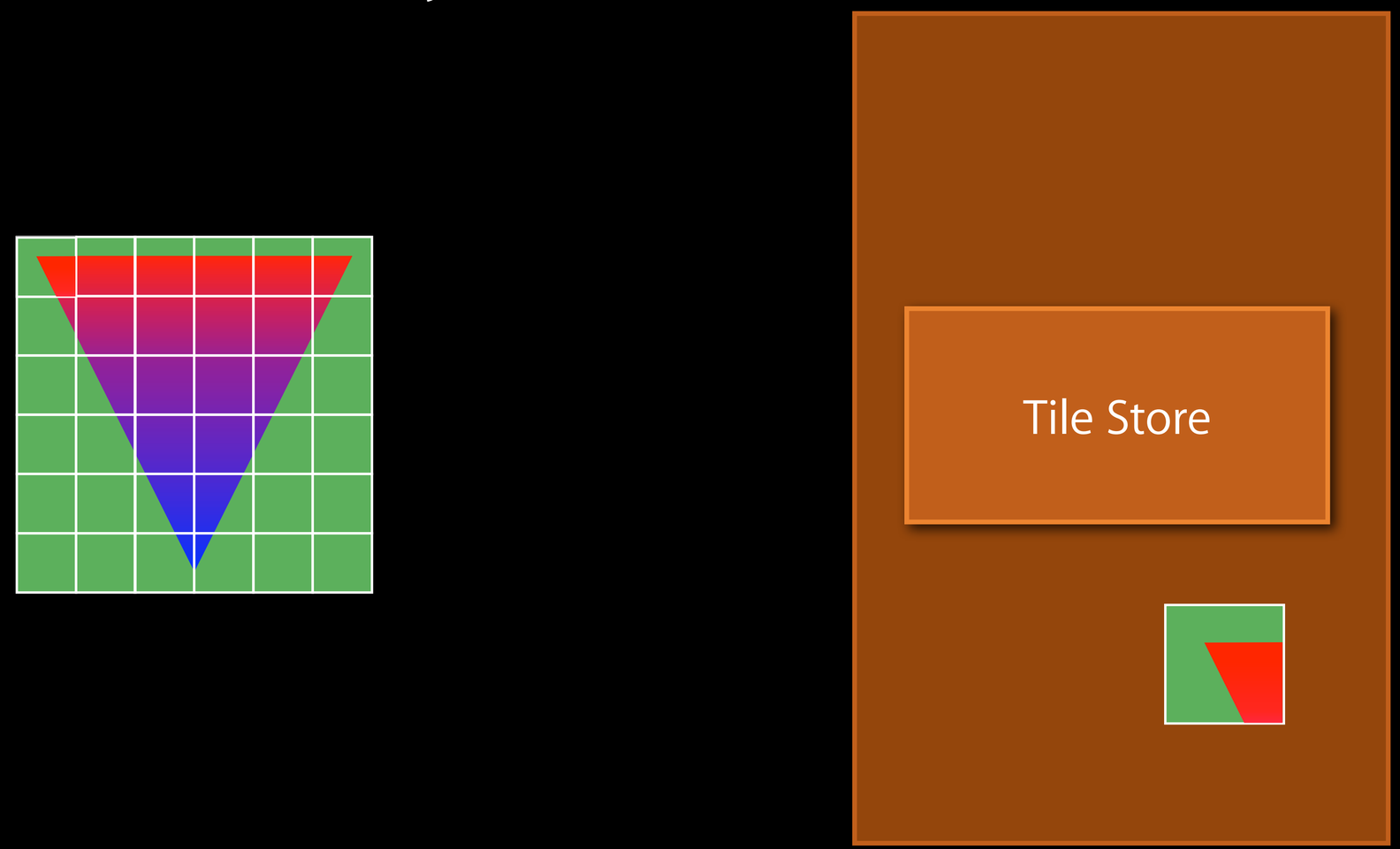
- Only need color buffer MSAA buffer has been resolved to



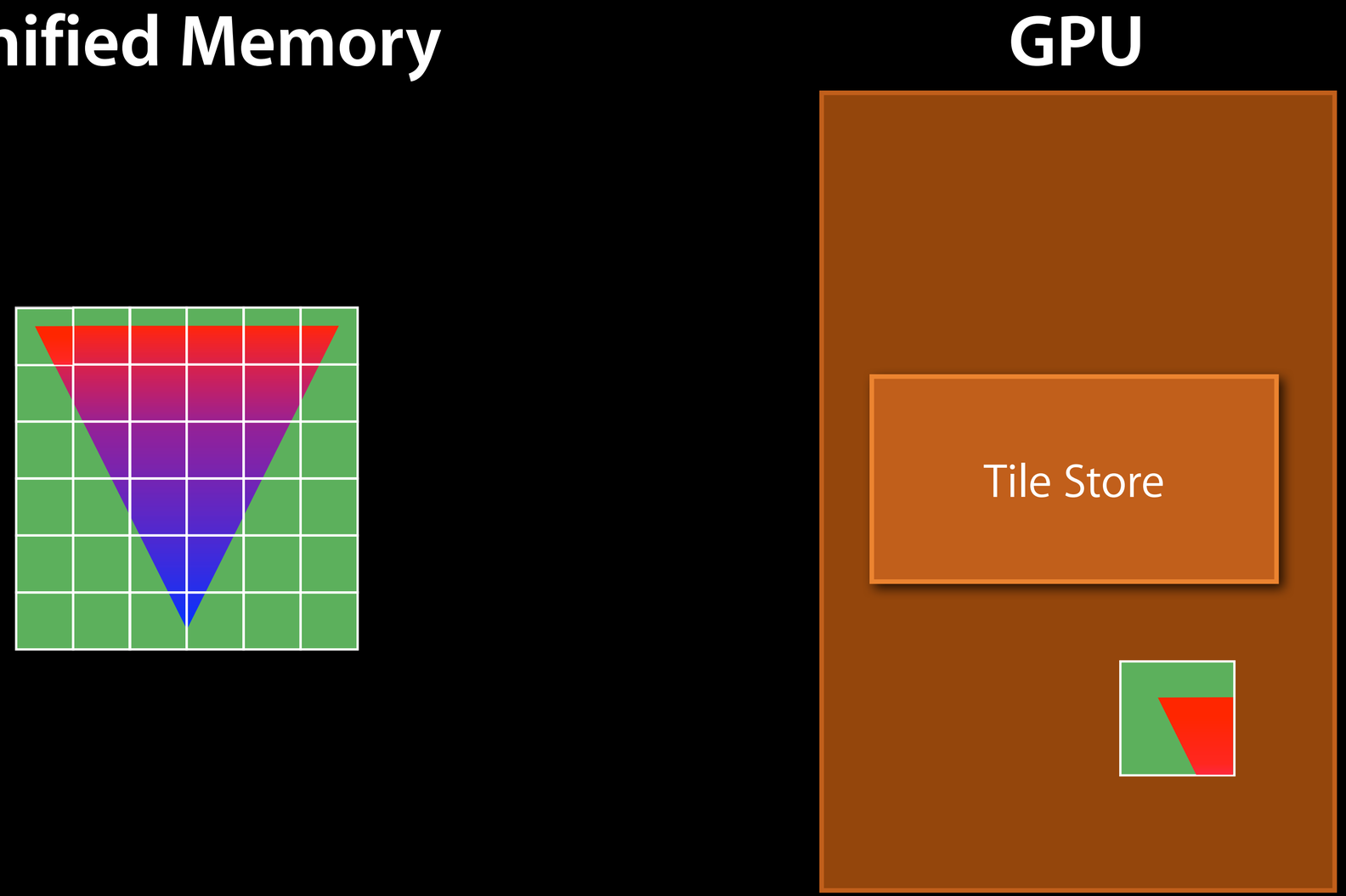
- Only need color buffer MSAA buffer has been resolved to



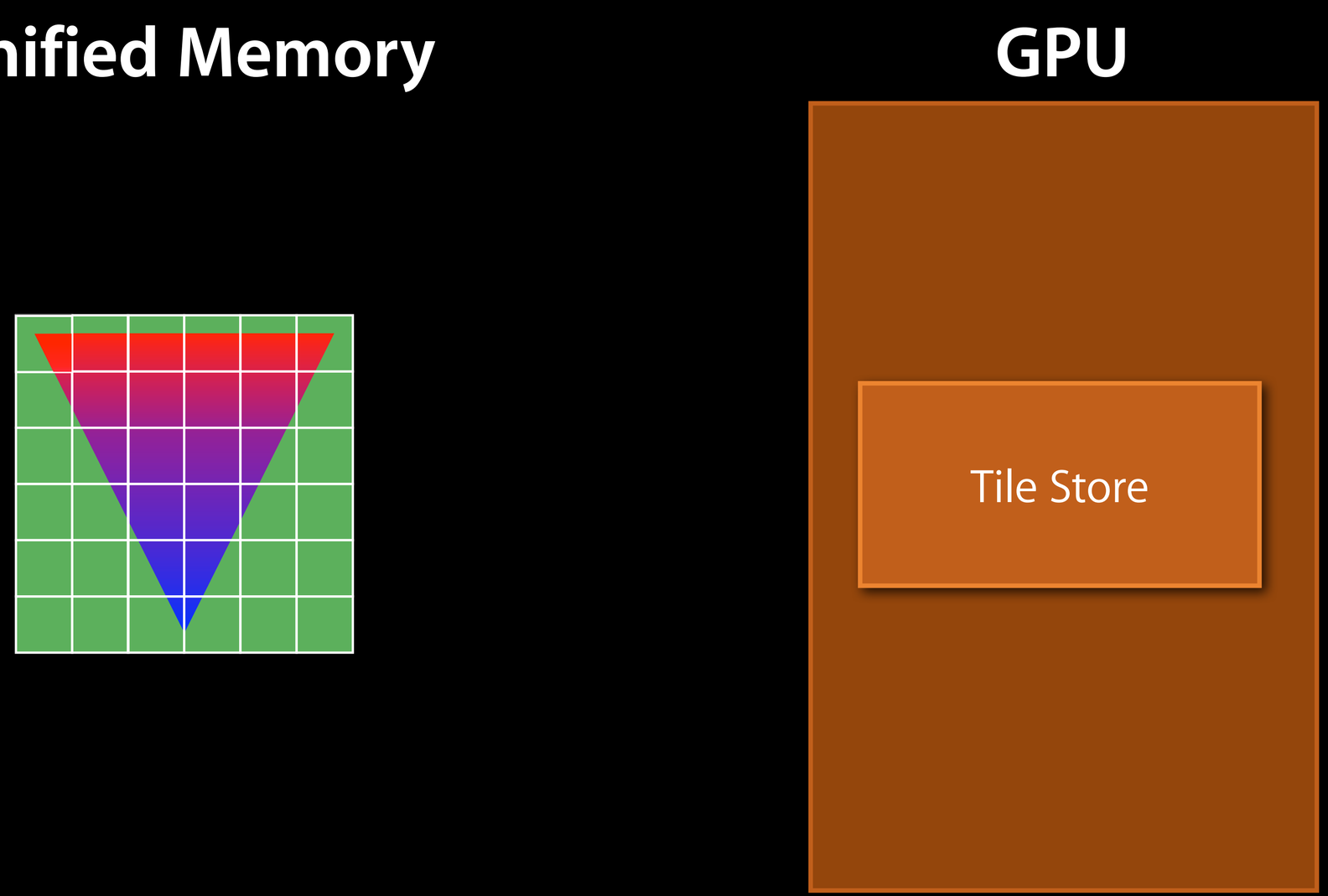
- Only need color buffer MSAA buffer has been resolved to



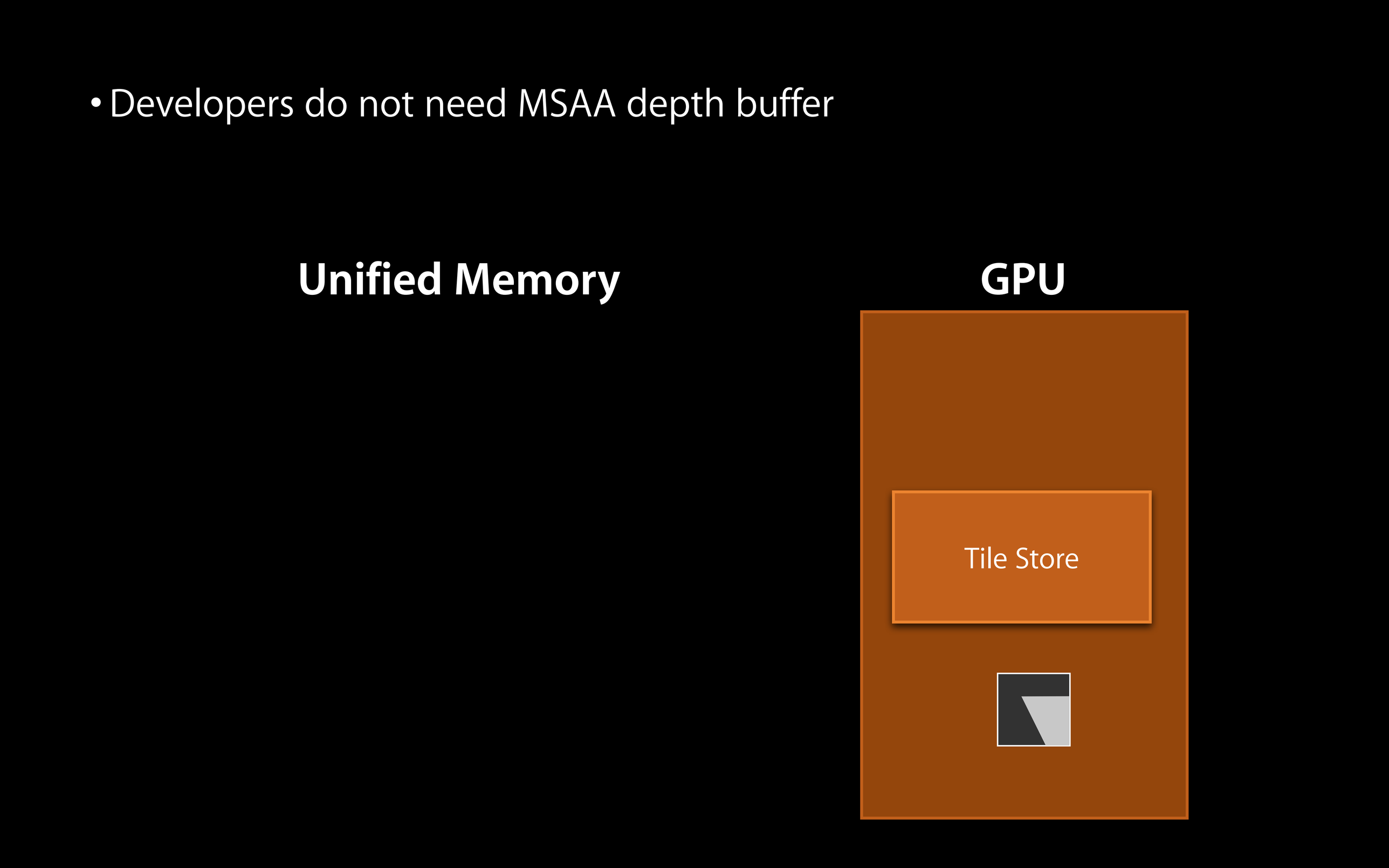
- Only need color buffer MSAA buffer has been resolved to
  - Call `glDiscardFramebufferEXT` on MSAA color buffer



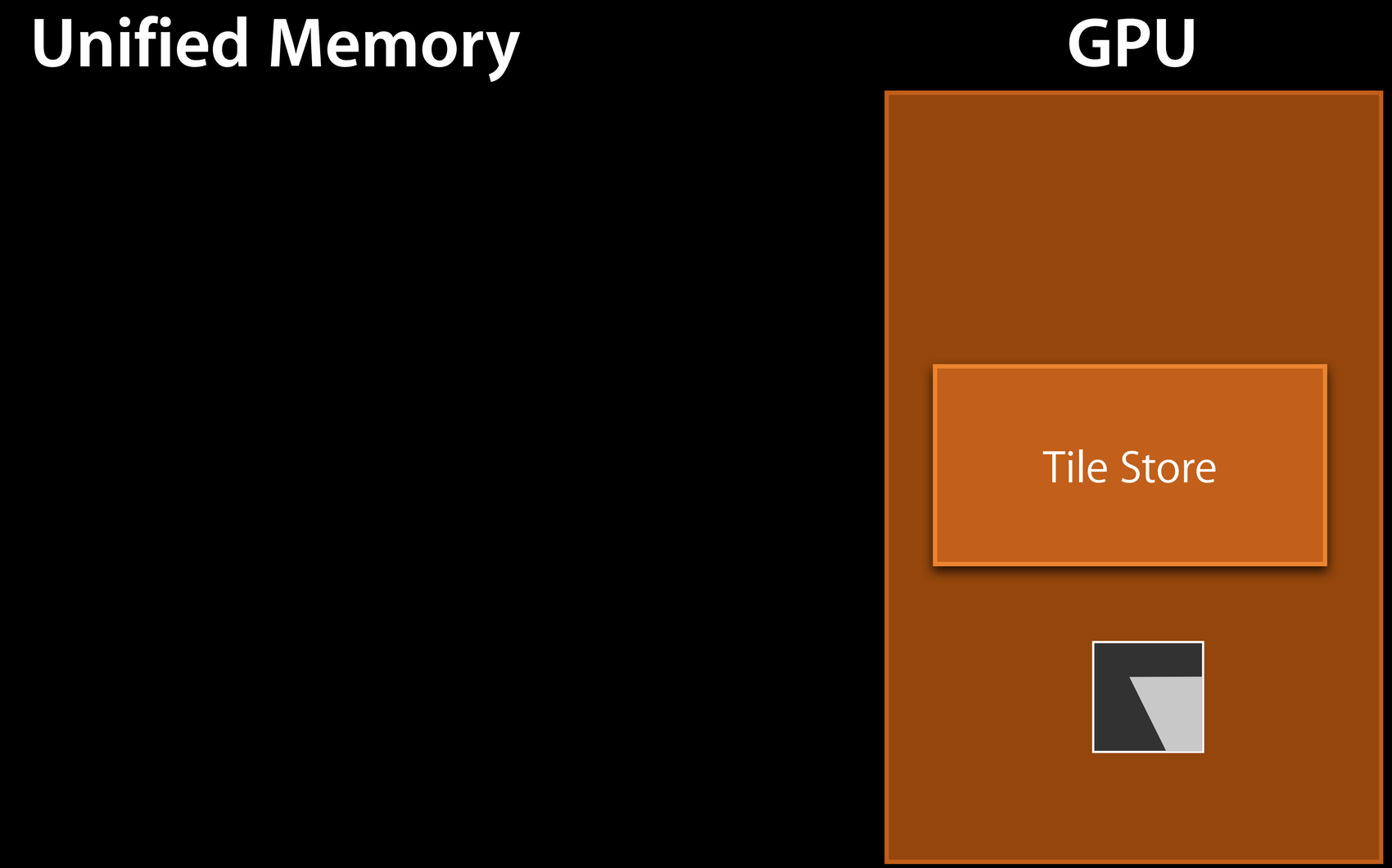
- Only need color buffer MSAA buffer has been resolved to
  - Call `glDiscardFramebufferEXT` on MSAA color buffer



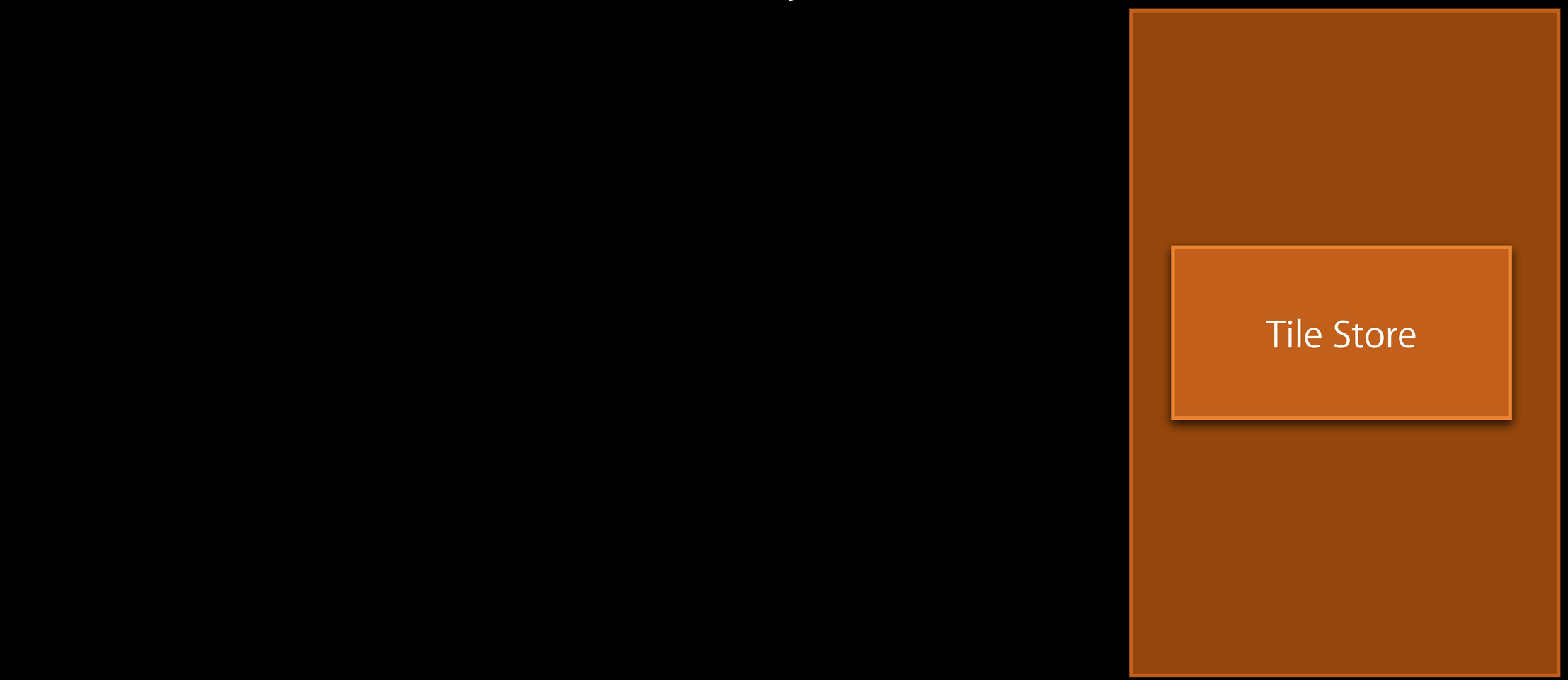
- Developers do not need MSAA depth buffer



- Developers do not need MSAA depth buffer
  - Also call `glDiscardFramebufferEXT` on MSAA depth buffer



- Developers do not need MSAA depth buffer
  - Also call `glDiscardFramebufferEXT` on MSAA depth buffer



Taking TBDR into Consideration

Hidden Surface Removal

- Hidden Surface Removal a unique strength
  - Greatly reduces workload
  - Certain operation defeat HSR process
    - Enabling blending
    - Using `discard` in shader

Taking TBDR into Consideration

Using fragment discard and blending

- Draw all triangles using `discard` after drawing triangles that do not
  - Hidden Surface Removal used for triangles in opaque group

Taking TBDR into Consideration

Using fragment discard and blending

- Draw all triangles using `discard` after drawing triangles that do not
  - Hidden Surface Removal used for triangles in opaque group
- Trim geometry needing these operations
  - Worth adding more vertices to reduce fragments

Taking TBDR into Consideration

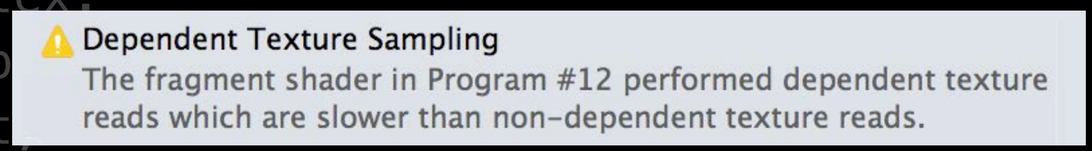
Logical Buffer Loads and Stores

- Transfers buffer between memory and GPU are expensive
- Avoid Logical Buffer Loads
  - Use `glClear` so GPU can skip them
  - Frequent renderbuffer switches cause tile thrashing
- Avoid Logical Buffer Stores
  - Use `glDiscardFramebufferEXT`
  - Especially for multisample anti-aliased buffers

Dependent Texture Sampling

- Calculate texture coordinate in shader
  - Sample with `texture*` function

```
uniform sampler2D tex;
varying vec2 texCoo
varying vec2 offset;
...

main()
{
    vec2 offsetCoord = texCoord + offset;
    vec4 color = texture(tex, offsetCoord);

    ...

}
```



Dependent Texture Sampling

- Calculate texture coordinate in shader
  - Sample with `texture*` function

```
uniform sampler2D tex;
varying vec2 texCoord;
varying vec2 offset;
...

main()
{
    vec2 offsetCoord = texCoord + offset;
    vec4 color = texture(tex, offsetCoord);

    ...

}
```

Dependent Texture Sampling

- Calculate texture coordinate in shader
  - Sample with `texture*` function

```
uniform sampler2D tex;
varying vec2 texCoord;
varying vec2 offset;
...

main()
{
    vec2 offsetCoord = texCoord + offset;
    vec4 color = texture(tex, offsetCoord);

    ...

}
```

Dependent Texture Sampling

- Calculate texture coordinate in shader
  - Sample with `texture*` function

```
uniform sampler2D tex;
varying vec2 texCoord;
varying vec2 offset;
...

main()
{
    vec2 offsetCoord = texCoord + offset;
    vec4 color = texture(tex, offsetCoord);

    ...

}
```

Dependent Texture Sampling

- Calculate texture coordinate in shader
  - Sample with `texture*` function

```
uniform sampler2D tex;
varying vec2 texCoord;
varying vec2 offset;
...

main()
{
    vec2 offsetCoord = texCoord + offset;
    vec4 color = texture(tex, offsetCoord);

    ...

}
```

Dependent Texture Sampling

- Calculate texture coordinate in shader
  - Sample with `texture*` function

```
uniform sampler2D tex;
varying vec2 texCoord;
varying vec2 offset;
...

main()
{
    vec2 offsetCoord = texCoord + offset;
    vec4 color = texture(tex, offsetCoord);

    ...

}
```

Dependent Texture Sampling

```
uniform sampler2D tex;
varying vec4 packedTexCoords;

...

main()
{
    vec4 color1 = texture2D(tex, packedTexCoords.xy);
    vec4 color2 = texture2D(tex, packedTexCoords.zw);
    ...

}
```

Dependent Texture Sampling

```
uniform sampler2D tex;
varying vec4 packedTexCoords;

...

main()
{
    vec4 color1 = texture2D(tex, packedTexCoords.xy);
    vec4 color2 = texture2D(tex, packedTexCoords.zw);
    ...

}
```

Dependent Texture Sampling

```
uniform sampler2D tex;
varying vec4 packedTexCoords;

...

main()
{
    vec4 color1 = texture2D(tex, packedTexCoords.xy);
    vec4 color2 = texture2D(tex, packedTexCoords.zw);

    ...

}
```

Dependent Texture Sampling

```
uniform sampler2D tex;
varying vec4 packedTexCoords;

...

main()
{
    vec4 color1 = texture2D(tex, packedTexCoords.xy);
    vec4 color2 = texture2D(tex, packedTexCoords.zw);

    ...

}
```

Dependent Texture Sampling

```
uniform sampler2D tex;
varying vec4 packedTexCoords;

...

main()
{
    vec4 color1 = texture2D(tex, packedTexCoords.xy);
    vec4 color2 = texture2D(tex, packedTexCoords.zw);
    ...

}
```

Dependent Texture Sampling

- High latency to sample texture in unified memory
  - GPU prefetches texture data for nondependent reads
    - Done in rasterization before shader executed
  - Cannot prefetch if coordinate calculated in shader
    - Shader stalls, waiting for texture data
- Minimize dependent texture samples
  - Hoist calculation
    - Perform coordinate calculation in vertex shader
    - Calculate in app and put in uniform

Dependent Texture Sampling

```
uniform sampler2D tex;
varying vec2 texCoord0;
varying vec2 texCoord1;

...

main()
{
    vec4 color1 = texture2D(tex, texCoord0);
    vec4 color2 = texture2D(tex, texCoord1);

    ...

}
```

Dependent Texture Sampling

```
uniform sampler2D tex;
varying vec2 texCoord0;
varying vec2 texCoord1;

...

main()
{
    vec4 color1 = texture2D(tex, texCoord0);
    vec4 color2 = texture2D(tex, texCoord1);

    ...

}
```

Dependent Texture Sampling

```
uniform sampler2D tex;
varying vec2 texCoord0;
varying vec2 texCoord1;

...

main()
{
    vec4 color1 = texture2D(tex, texCoord0);
    vec4 color2 = texture2D(tex, texCoord1);

    ...

}
```

Dependent Texture Sampling

```
uniform sampler2D tex;
varying vec2 texCoord0;
varying vec2 texCoord1;

...

main()
{
    vec4 color1 = texture2D(tex, texCoord0);
    vec4 color2 = texture2D(tex, texCoord1);

    ...

}
```

Dynamic Branching

A simple example

```
attribute float testVal;
varying float outVal;

...

main()
{
```

⚠ Fragment Shader Dynamic Branching
The fragment shader in Program #14 "Light" contains dynamic branch instructions, which reduces the performance of the shader.

```
    {
        outVal = 1.0;
    }
    else
    {
        outVal = 0.0;
    }

    ...

}
```

Dynamic Branching

A simple example

```
attribute float testVal;
varying float outVal;

...

main()
{
    if(testVal > 1.0)
    {
        outVal = 1.0;
    }
    else
    {
        outVal = 0.0;
    }

    ...
}
```

Dynamic Branching

A simple example

```
attribute float testVal;
varying float outVal;

...

main()
{
    if(testVal > 1.0)
    {
        outVal = 1.0;
    }
    else
    {
        outVal = 0.0;
    }

    ...
}
```

Dynamic Branching

A simple example

```
attribute float testVal;
varying float outVal;

...

main()
{
    if(testVal > 1.0)
    {
        outVal = 1.0;
    }
    else
    {
        outVal = 0.0;
    }

    ...

}
```

Dynamic Branching

A simple example

```
attribute float testVal;
varying float outVal;

...

main()
{
    if(testVal > 1.0)
    {
        outVal = 1.0;
    }
    else
    {
        outVal = 0.0;
    }

    ...

}
```

Dynamic Branching

A simple example

```
attribute float testVal;
varying float outVal;

...

main()
{
    if(testVal > 1.0)
    {
        outVal = 1.0;
    }
    else
    {
        outVal = 0.0;
    }

    ...
}
```

Dynamic Branching

- GPUs are highly parallel devices
  - Process multiple vertices and fragments simultaneously
- Special branch mode for execution
  - More latency to stay in sync
- If possible, calculate predicate outside of shader
  - Branches on uniforms do not incur same overhead

Dynamic Branching

- GPUs are highly parallel devices
  - Process multiple vertices and fragments simultaneously
- Special branch mode for execution
  - More latency to stay in sync
- If possible, calculate predicate outside of shader
  - Branches on uniforms do not incur same overhead
- Shaders using both Dependent Texture Sampling and Dynamic Branching are particularly costly

Minimizing CPU Overhead

Managing draw calls

Optimizing Draw Call Performance

- Most CPU overhead in draw calls
- More state set for draw, more expensive draw
  - State setting looks inexpensive
  - Work for state set deferred until draw

Optimizing Draw Call Performance

- Most CPU overhead in draw calls
- More state set for draw, more expensive draw
  - State setting looks inexpensive
  - Work for state set deferred until draw
- Maximize efficiency of each draw

Optimizing Draw Call Performance

- Most CPU overhead in draw calls
- More state set for draw, more expensive draw
  - State setting looks inexpensive
  - Work for state set deferred until draw
- Maximize efficiency of each draw

Optimizing Draw Call Performance

- Most CPU overhead in draw calls
- More state set for draw, more expensive draw
  - State setting looks inexpensive
  - Work for state set deferred until draw

⚠ Redundant Call
glBindTexture(GL_TEXTURE_2D, 1u) set a piece of GL state to its current value.

- Maximize efficiency of each draw

⚠ Inefficient State Update
glViewport(0, 0, 1024, 1024) sets a piece of GL state more than once before it is used by a clear or a draw call.

Optimizing Draw Call Performance

- Most CPU overhead in draw calls
- More state set for draw, more expensive draw
  - State setting looks inexpensive
  - Work for state set deferred until draw
- Maximize efficiency of each draw
- Reduce inefficient state setting
  - Shadow state
  - Sort state

Optimizing Draw Call Performance

- Some fixed overhead for draw
  - State validation
  - Call to driver

Optimizing Draw Call Performance

- Some fixed overhead for draw
  - State validation
  - Call to driver
- Minimize number of draw calls made
  - Object culling
  - Coalesce calls
    - Instancing
    - Vertex batching
    - Texture atlases

Texture Atlases

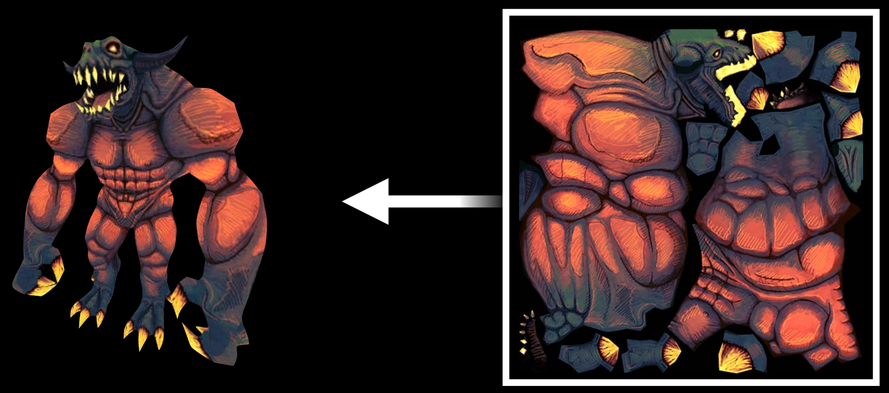
Bind and draw

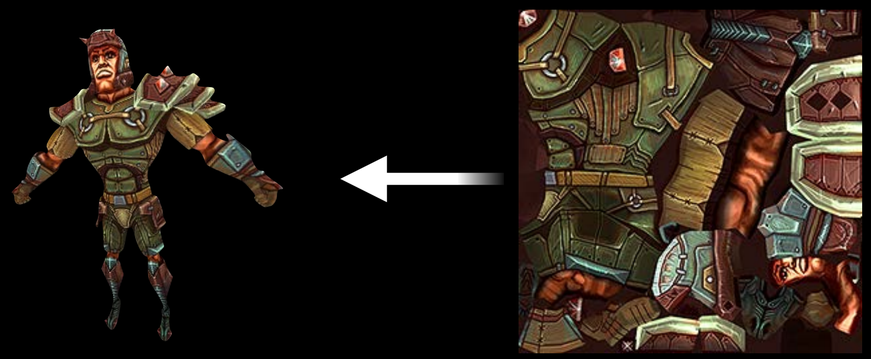
Texture Atlases

Bind and draw

Texture Atlases

Bind and draw



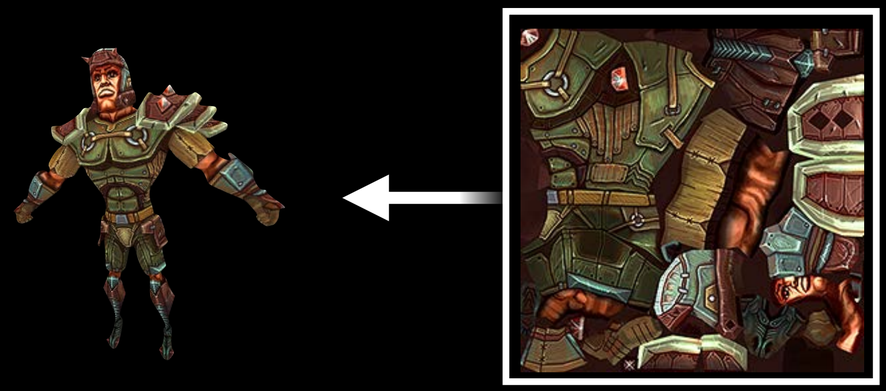


Texture Atlases

Bind and draw

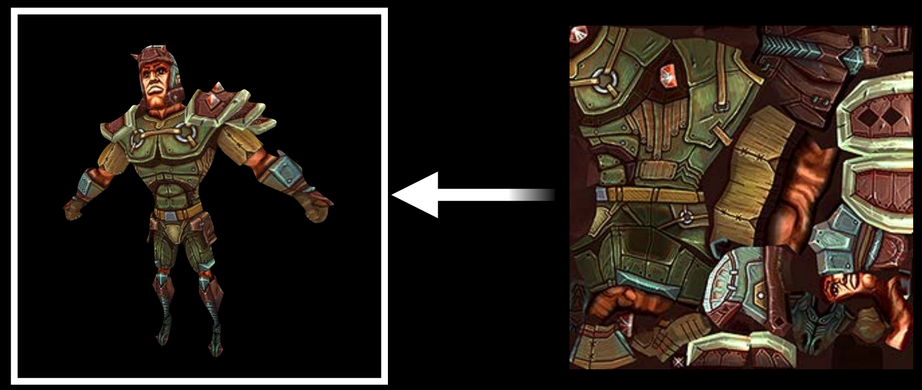
Texture Atlases

Bind and draw



Texture Atlases

Bind and draw



Texture Atlases

Bind and draw

Texture Atlases

Bind and draw

Texture Atlases

Bind and draw

Texture Atlases

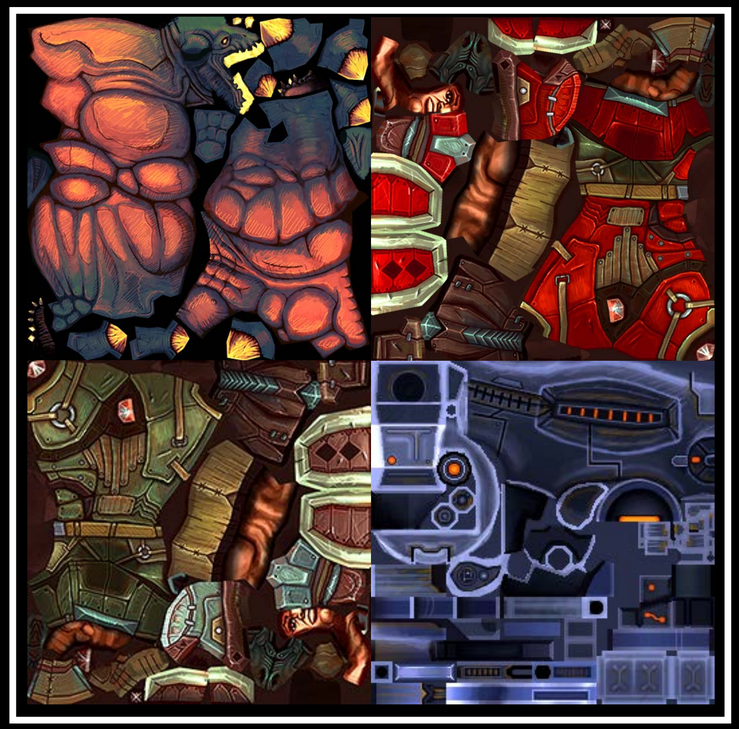
Bind and draw

Texture Atlases

Reducing binds

Texture Atlases

Reducing binds

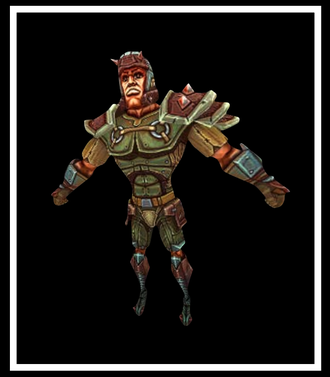


Texture Atlases

Reducing binds

Texture Atlases

Reducing binds



Texture Atlases

Reducing binds

Texture Atlases

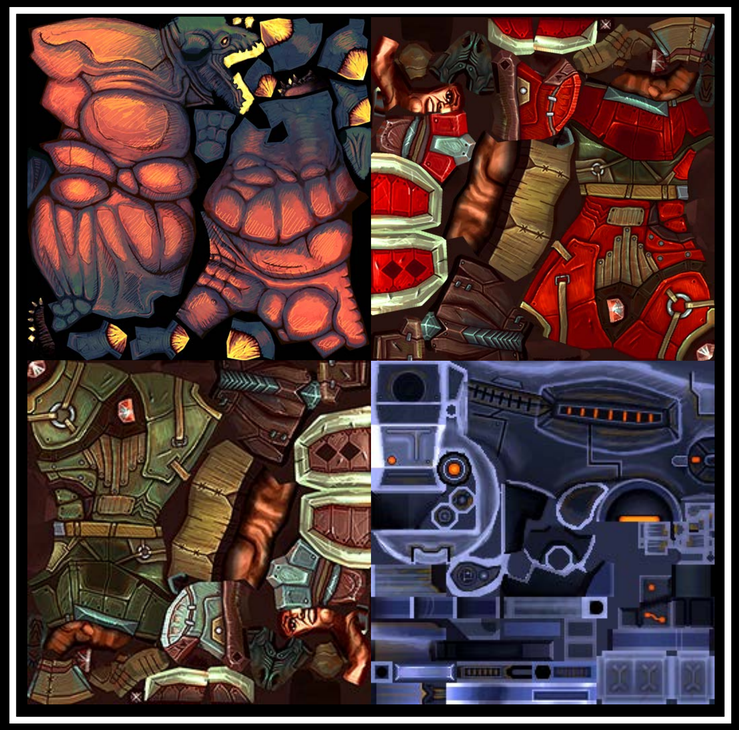
Reducing binds

Texture Atlases

Combining draws

Texture Atlases

Combining draws



Texture Atlases

Combining draws

Texture Atlases

Sprite Kit texture atlas tool

- Combines images efficiently
- Produces property list denoting sub-images
  - Scale texture coordinates based on plist
- ‘TextureAtlas’ command line tool in Xcode

More Information

Allan Schaffer
Graphics and Game Technologies Evangelist
aschaffer@apple.com

Documentation
OpenGL ES Programming Guide for iOS
https://developer.apple.com/opengles

Apple Developer Forums
http://devforums.apple.com

Related Sessions

Introduction to Sprite Kit	Presidio Wednesday 11:30AM
What’s New in OpenGL for OS X	Marina Thursday 2:00PM

Labs

Sprite Kit Lab	Graphics and Games Lab B Thursday 9:00AM
OpenGL and OpenGL ES Lab	Graphics and Games Lab A Thursday 10:15AM

Summary

- Reduce draw call overhead
  - Use techniques including instancing and texture atlases
- Consider GPU's operation when architecting your renderer and in performance investigations
  - GPU tools help greatly
  - Tile-Based Deferred Rendering has some special consideration

WWDC2013