U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Render to Texture

Concepts and Practice

Teresa Matos
07 November 2019

Contents

Graphics Pipeline

- Inputs
- Vertex shading
- Primitive assembly
- Geometry shading
- Projection and **rasterization**
- Fragment shading
- Raster operations
- Output to screen

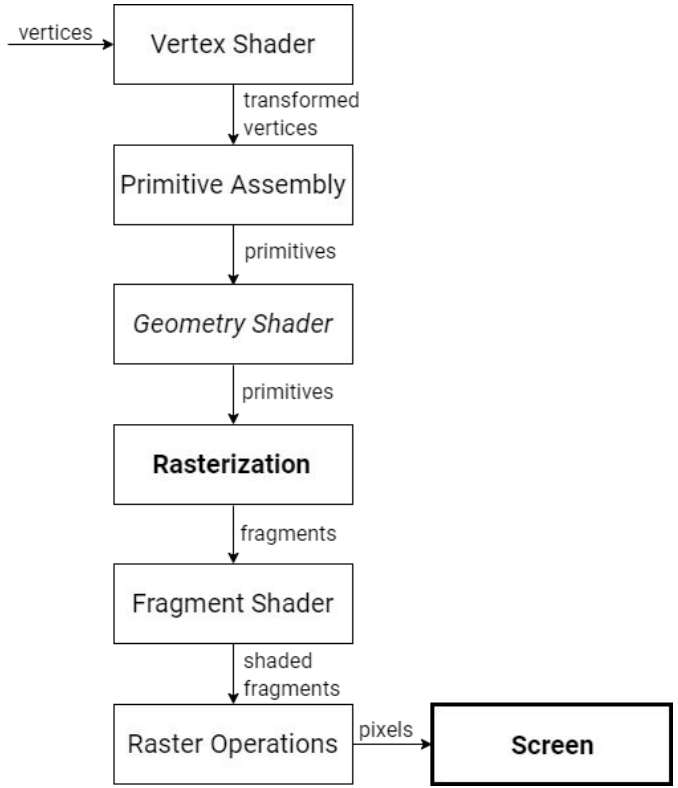


Fig. 1: OpenGL simplified pipeline (adapted from [1])

Rasterization

Receives list of **assembled primitives**

Defines list of **fragments** - collection of values for every pixel

Saves fragments to **default framebuffer**

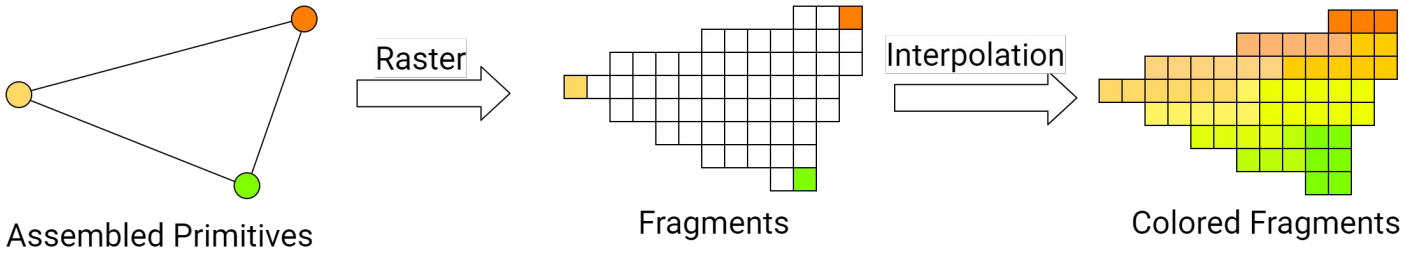


Fig. 2: Rasterization process - Fragments are obtained for the pixels that cover the assembled primitives [1]

Framebuffer

Buffer that holds **color data** of an image, stored in the video RAM

Composed by **color**, **depth**, and **stencil** buffers

Several framebuffers may be used at the same time, for:

- Double buffering for animation
- **Render to texture**

Render to Texture

Scene is rendered into a texture, instead of canvas

Applications of render to texture (RTT):

- Reflection mapping
- Shadow mapping
- Post-processing effects
- **In-scene cameras**

Render to Texture: Reflection Mapping

Reflection on objects is simulated with **pre-rendered environment**

Environment is rendered to texture, and **mapped to an object surrounding** the reflective object, e.g., an inverted sphere or cube

Less expensive than raytracing

No reflection between objects
or in objects with convex surfaces

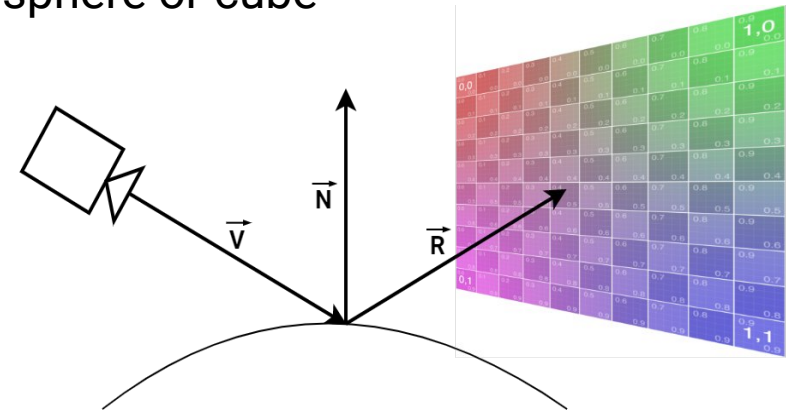


Fig. 3: Reflection mapping - Reflection vector, obtained from normal and view vectors, is used to obtain texel from environment map

Render to Texture: Shadow mapping

Scene is rendered from **light source**

Visibility is translated to **lit/shadowed fragments**

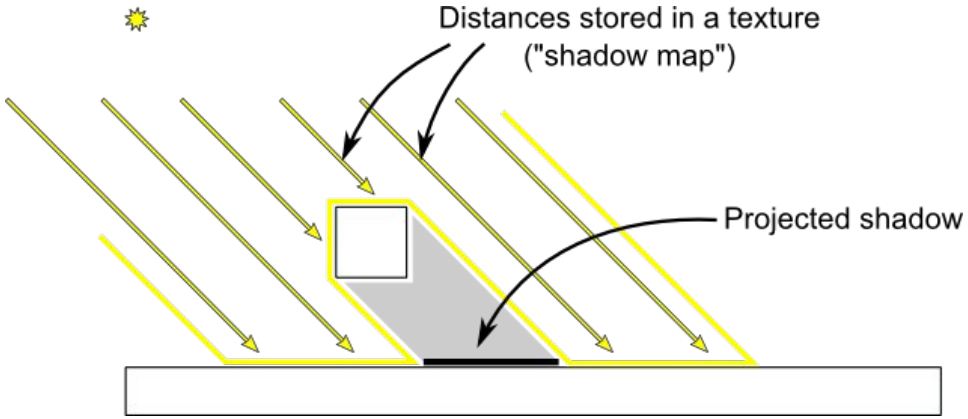


Fig. 4: Shadow mapping - Scene is rendered from light, visible areas (in yellow) are lit and hidden areas (in grey) are in shadow [2]

Render to Texture: Shadow mapping (2)

Depth buffer is saved in texture - **depth/shadow map**

Fragment shader applies shadow map on fragments

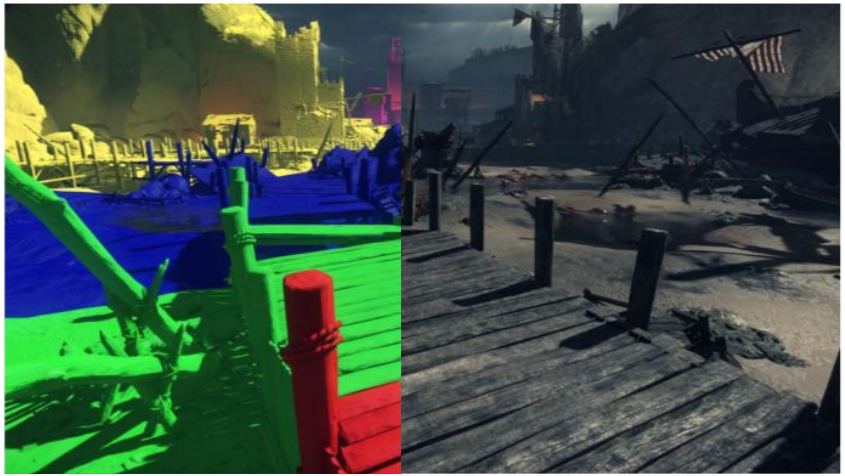


Fig. 4: Cascaded shadow mapping, several shadow maps are created according to distance to viewer [3]

Render to Texture: Post-processing

Render scene or objects to texture

Apply **post-processing effects** (i.e., blur) using shaders

Fill screen with processed texture,
or **map to object**

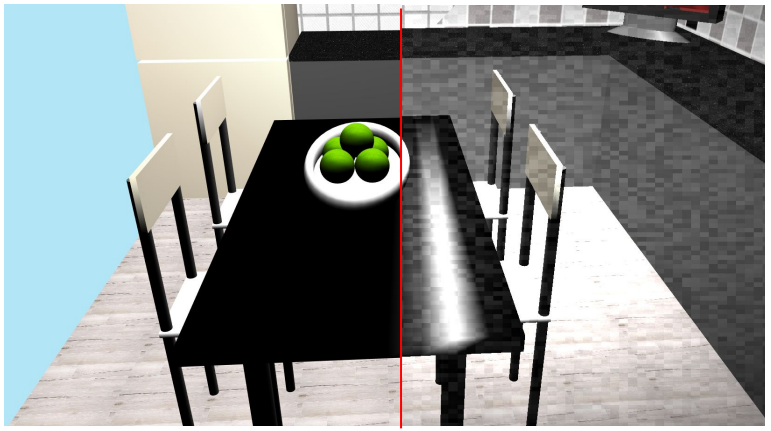


Fig. 5: Post-processing: Appearance of rendered scene to the left, and rendered to texture with noise effects to the right

Render to Texture: In-scene Cameras

Render scene from different perspectives to texture

Useful for **bird's eye view**, **split screens**, and others

Fig. 6: Split screen for different players in Rocket League [4]

Render to Texture: Process

1. Create target texture and framebuffer
2. Render scene to target
3. Apply texture

Render to Texture: Process

1. Create target texture and framebuffer
   - Create 2D texture without allocated image
   - Create framebuffer (optional depth buffer)
   - Bind 2D texture to framebuffer
2. Render scene to target
3. Apply texture

Render to Texture: Process

1. Create target texture and framebuffer
2. Render scene to target
   - Bind target framebuffer
   - Render scene to texture
   - Unbind target framebuffer (return to default)
   - Render scene to canvas
3. Apply texture

Render to Texture: Process

1. Create target texture and framebuffer
2. Render scene to target
3. Apply texture
    - Apply shader with additional effects (optional)
    - Bind texture
    - Draw object with applied texture

Render to Texture in WebCGF

CGFtextureRTT Class

1. Handles creation of target texture and framebuffer
2. Provides functions to bind/unbind framebuffer
3. Provides functions to bind/unbind texture

***CGFtextureRTT**(scene, width, height)*

Render to Texture Example: Security Camera

1. Render scene to *CGFtextureRTT* texture using different camera
2. Render scene to canvas
3. Apply *CGFtextureRTT* texture to UI object - display
4. Add effects to UI object using shaders

Security Camera: Changes to CGFscene

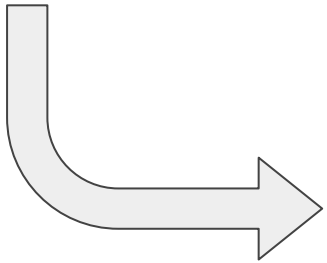
display:

Setup background

Setup scene buffers

Update lights

Display scenegraph



display:

Attach CGFtextureRTT framebuffer

Call render() with RTT camera

Detach CGFtextureRTT framebuffer

Call render() with scene camera

render:

Setup background

Setup scene buffers

Set requested camera

Update lights

Display scenegraph

Security Camera: UI object

Create *MyRectangle* object to be displayed as UI object

Define vertices that correspond to the bottom right corner of screen

Apply vertex shader to draw object in screen

```
gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
```

Projection matrix

View matrix

Model matrix

Removing these matrices draws the object in screen

Security Camera: Post-processing effects

Apply fragment shader to drawn object

1. Add gradient to texture

Example: Linear gradient

Multiply texture color by horizontal component of texture coordinates:

x = 0.0 => black color

x = 1.0 => pure color from texture

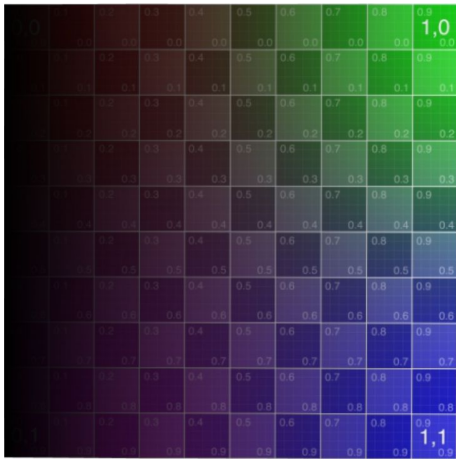


Fig. 7: Quad with horizontal linear gradient applied to texture

```
fragColor = vec4(color.rgb * vTextureCoord.x, 1.0);
```

Color from texture (uSampler)

Security Camera: Post-processing effects

Apply fragment shader to drawn object

2. Add animated horizontal lines to texture

Pass time to shader as uniform value

Example: Add horizontal lines with 10% height

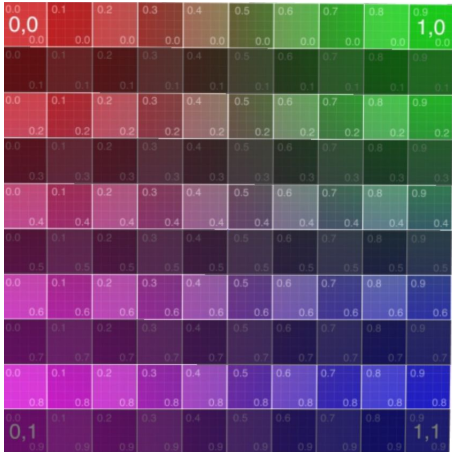


Fig. 8: Quad with horizontal stripes applied to texture

Varies from [0.0, 2.0[and loops 5 times for texCoord.y ∈ [0.0, 1.0]

```
if(mod(vTextureCoord.y * 10.0, 2.0) > 1.0)
    color = vec4(color.rgb*0.5,1.0);

fragColor = vec4(color.rgb, 1.0);
```

Darkening color from texture (uSampler)

References

1. Fernandes, A. (2011). Pipeline Overview. LightHouse3D Tutorials (accessed November 2019). (http://www.lighthouse3d.com/tutorials/glsl-12-tutorial/pipeline-overview/)
2. Tutorial 16 : Shadow mapping (2016). OpenGL Tutorials (accessed November 2019). (http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping)
3. Kasyan, N. (2013). Playing with real-time shadows. *SIGGRAPH'13: ACM SIGGRAPH 2013 Courses*. (https://www.realtimeshadows.com/)
4. Sampaio, C. (2016). How to Get Rocket League Splitscreen to Work on PC. *GameSkinny*. (https://www.gameskinny.com/jq95r/how-to-get-rocket-league-splitscreen-to-work-on-pc)
5. WebGL Rendering to a Texture (2017). *WebGL Fundamentals*. (https://webglfundamentals.org/webgl/lessons/webgl-render-to-texture.html)