Abertay University®

Render to Texture / Render targets

CMP301 Graphics Programming with Shaders

This week

- Render to texture
  - What is it
  - What is it for
  - How
  - Code / example

Render to texture

- Currently we render directly to the back buffer / frame buffer
- This data is then pushed to the screen via another buffer
- The back buffer is essentially a texture
  - A 2D collection of colour data
- Therefore why not render to something that isn’t the back buffer?

Render to texture

- What is it
  - The ability to render the scene (or objects) to a 2D image (texture) instead of the back buffer
- We are giving the pipeline another “render target”
- This allows further processing of the scene
  - Render it normally to a texture
  - Then render it again to the back buffer
    - Using another shader to achieve unique effects

Render to texture

- Purposes
    - Shadow mapping
    - Post Processing
    - Dynamic reflection
    - Used in deferred rendering
    - Core of multi-pass rendering
- Example
    - Render to texture the scene from a bird’s eye view
    - Draw the scene normally
    - Draw a quad using the texture of the bird’s eye view
    - Creating a simple radar / mini map system

Render to texture

- This is a seriously powerful tool
- But VERY expensive (resource wise)
  - May be rendering parts of the scene multiple times

Render to texture

- This requires a render target view
    - ID3D11RenderTargetView
    - Linking a shader resource view to it
        - A texture
    - Bind the new render target to the OM (output merger) stage of the pipeline
        - OMSetRenderTarget()
- Also require a method for swapping the render target to the back buffer

Render to texture

- I have provided a class which encapsulates the render to texture
    - RenderTexture()
        - Creates the render target
        - Texture
        - Other required objects

```
renderTexture = new RenderTexture(renderer->getDevice(), screenWidth,
screenHeight, SCREEN_NEAR, SCREEN_DEPTH);
```

Render to texture

- Other functions
  - getShaderResourceView ()
    - Returns the texture
    - Needed when re-rendering the object
  - setRenderTarget ()
    - Changes the render target to this object
    - Can have more than one
  - clearRenderTarget ()
    - Blank render to texture with provided colour
- There are others functions, but not worth noting here

Output

- We need someway of displaying our render to texture
  - A simple object we can render with our created texture
    - Could use a simple mesh quad
  - Going to use OrthoMesh (in the framework)
    - Is a simple quad
    - Built for render to texture processing
    - Can be sized and positioned in the window
    - For orthographic rendering (GUI like)

Orthographic projection

- Render objects won’t shrink when further away from the camera
  - Unlike the normal projection we do
- Uses an axis aligned box, that looks down the Z+ axis
- Allows 2D rendering
- Very useful for GUI elements, debugging info and text
- Requires a different projection matrix when rendering

```
// ortho matrix for 2D rendering
orthoMatrix = renderer->getOrthoMatrix();
```

Orthographic projection

- Requires we turn off the Z buffer
- When we render the render-to-texture object we use the orthographic projection matrix

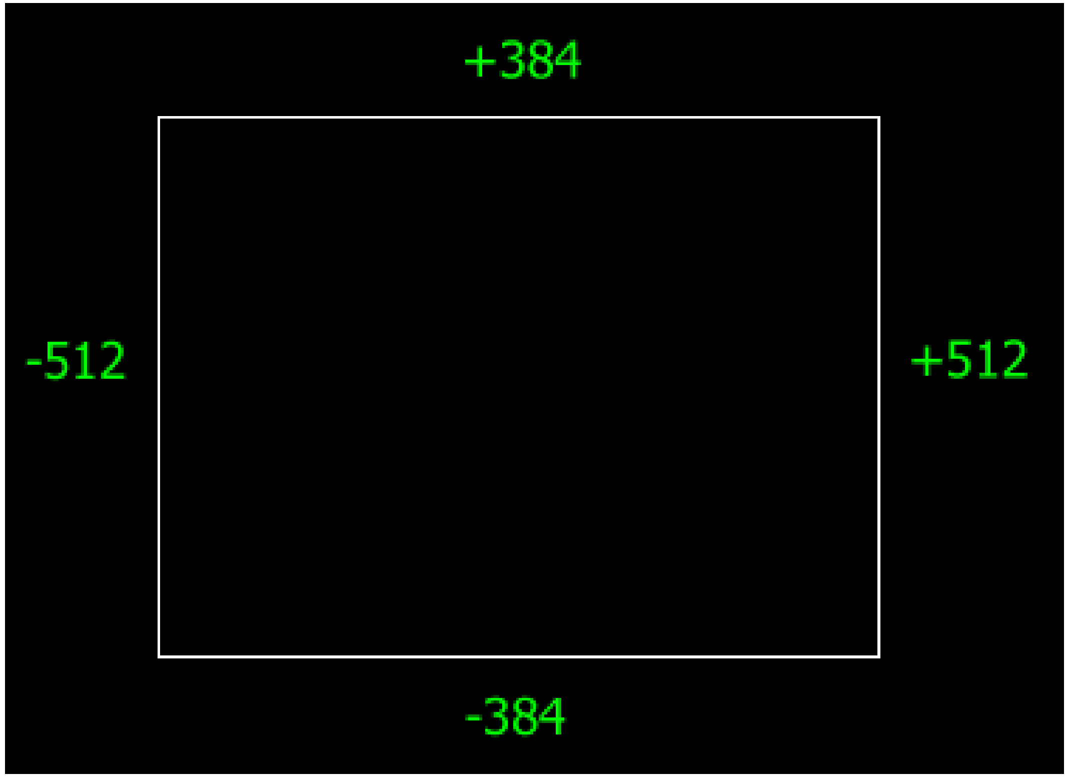
- Also requires another view matrix
  - A camera untransformed
  - Looking down the Z axis
  - orthoViewMatrix = camera->getBaseViewMatrix();

Ortho Mesh

- Simple modified mesh class
- Draws a quad
- Quad size is based on provided parameters
  - Meaning we can control it’s size
- Quad position can also be controlled
  - Designed for outputting a quad the size of the window
  - But we can control the position and size of the quad
- In the future we will be using the full screen version
  - But for now we need to see how the Render Target works

Ortho Mesh

- Screen coordinates have an origin at the centre
  - To position the quad -(screen width / 2)



Ortho Mesh

- OrthoMesh()
  - width and height
  - X, Y position
  - orthoMesh = new OrthoMesh(renderer->getDevice(), 200, 150, -300, 225);
- Uses the width, height, x and y to build a quad of the size and position specified
- Other than that functions like any other mesh
  - But has no texture
  - The texture will come from the RenderTexture/target

Ortho Mesh

- When rendering a orthoMesh
- Use a texture shader
  - We don’t require lighting
  - Different view matrix and use orthographic matrix
  - Texture is our renderTexture
  - E.g.

```
orthoMesh->sendData(renderer->getDeviceContext());

textureShader->setShaderParameters(renderer->getDeviceContext(), worldMatrix, orthoViewMatrix, orthoMatrix, renderTexture->getShaderResourceView());

textureShader->render(renderer->getDeviceContext(), orthoMesh->getIndexCount());
```

Example

- Keep it simple
- Scene will contain a simple mesh morphing example from last week
  - We will render this to texture
  - As well as displaying the scene as normal
  - We will display the renderTexture in a target window in the top left

Example

- Multi pass rendering
  - First pass
    - Render scene to renderTexture/target
  - Second pass
    - Render scene to back buffer
    - Additional orthographic quad to display the renderTexture
- Highly recommend separate functions for separate passes!

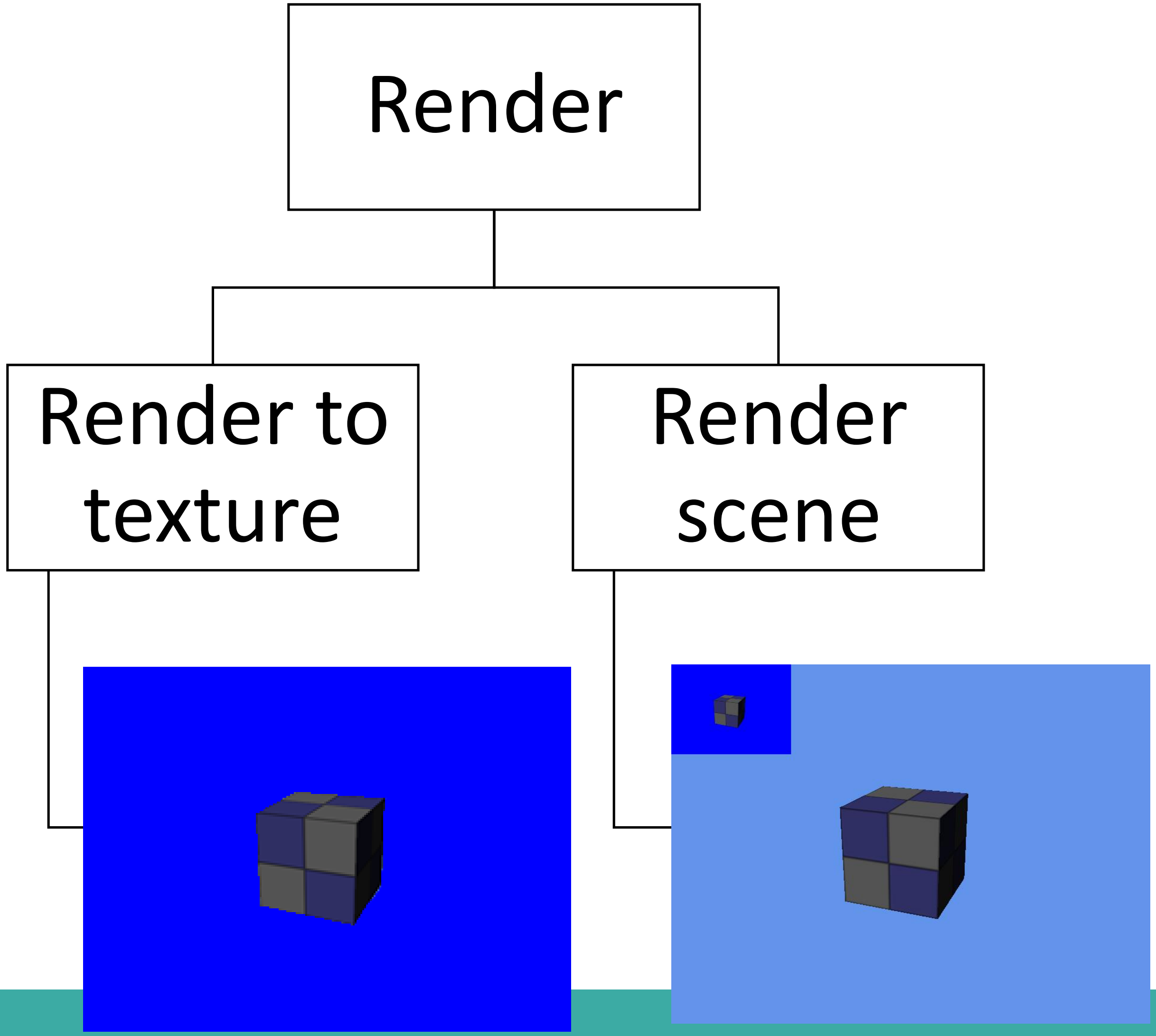
Example

- RenderToTexture()
    - Change the render target
    - Clear the render target (note I will set it to blue)
    - Render the scene
        - Instead of going onto the back buffer like normal it is rendered to the texture target
    - Reset the render target

Example

- RenderScene()
  - This goes on the back buffer
  - Our normal scene rendering
    - In this example a simple cube with a texture and some lighting
  - Finally we render the orthoMesh
    - This is the 2D object that will contain the image from the renderTexture
    - This requires no Z buffering
      - So before target window is rendered disable z buffer
      - Enable after rendering
      - Notice the orthoMesh is being draw last

Diagram



Render

Render to texture

Render scene

Application setup

```cpp
// Mesh and shader set for normal scene rendering
mesh = new CubeMesh(renderer->getDevice(), renderer->getDeviceContext());
shader = new MeshMorphShader(renderer->getDevice(), hwnd);

// Create light source (for normal scene rendering)
light = new Light;
light->setAmbientColour(0.1f, 0.1f, 0.1f, 1.0f);
light->setDiffuseColour(1.0f, 1.0f, 1.0f, 1.0f);
light->setDirection(0.5f, -0.5f, 0.0f);

// RenderTexture, OrthoMesh and shader set for different renderTarget
renderTexture = new RenderTexture(renderer->getDevice(), screenWidth, screenHeight, SCREEN_NEAR,
SCREEN_DEPTH);
// ortho size and position set based on window size
// 200x200 pixels (standard would be matching window size for fullscreen mesh
// Position default at 0x0 centre window, to offset change values (pixel)
orthoMesh = new OrthoMesh(renderer->getDevice(), renderer->getDeviceContext(), 200, 150, -300, 225);
textureShader = new TextureShader(renderer->GetDevice(), hwnd);
```

Render to texture

```cpp
void App::RenderToTexture()
{
XMMATRIX worldMatrix, viewMatrix, projectionMatrix;

// Set the render target to be the render to texture.
renderTexture->setRenderTarget(renderer->GetDeviceContext());

// Clear the render to texture.
renderTexture->clearRenderTarget(renderer->getDeviceContext(), 0.0f, 0.0f, 1.0f, 1.0f);

// Generate the view matrix based on the camera's position.
camera->update();

// Get the world, view, and projection matrices from the camera and d3d objects.
worldMatrix = renderer->getWorldMatrix();
viewMatirx = camera->getViewMatrix();
projectionMatrix = renderer->getProjectionMatrix();
```

Render to texture

```
// Put the model vertex and index buffers on the graphics pipeline to prepare them for drawing.
mesh->sendData(renderer->getDeviceContext());


shader->setShaderParameters(renderer->getDeviceContext(), worldMatrix, viewMatrix, projectionMatrix,
mesh->getTexture(), light, timer->getTime());
// Render object (combination of mesh geometry and shader process
shader->render(renderer->getDeviceContext(), mesh->getIndexCount());


// Reset the render target back to the original back buffer and not the render to texture anymore.
renderer->setBackBufferRenderTarget();


}
```

Render scene

```
void App::RenderScene()
{
XMMATRIX worldMatrix, viewMatrix, projectionMatrix, orthoViewMatrix, orthoMatrix;

// Clear the scene. (default blue colour)
renderer->beginScene(0.39f, 0.58f, 0.92f, 1.0f);

// Generate the view matrix based on the camera's position.
camera->update();

// Get the world, view, projection, and ortho matrices from the camera and Direct3D objects.
worldMatrix = renderer->getWorldMatrix();
viewMatrix = camera->getViewMatrix();
projectionMatrix = renderer->getProjectionMatrix();

// Send geometry data (from mesh)
mesh->sendData(renderer->getDeviceContext());
// Set shader parameters (matrices and texture)
shader->setShaderParameters(renderer->getDeviceContext(), worldMatrix, viewMatrix,
projectionMatrix, mesh->getTexture(), light, timer->getTime());
// Render object (combination of mesh geometry and shader process
shader->render(renderer->getDeviceContext(), mesh->getIndexCount());
```

Render scene

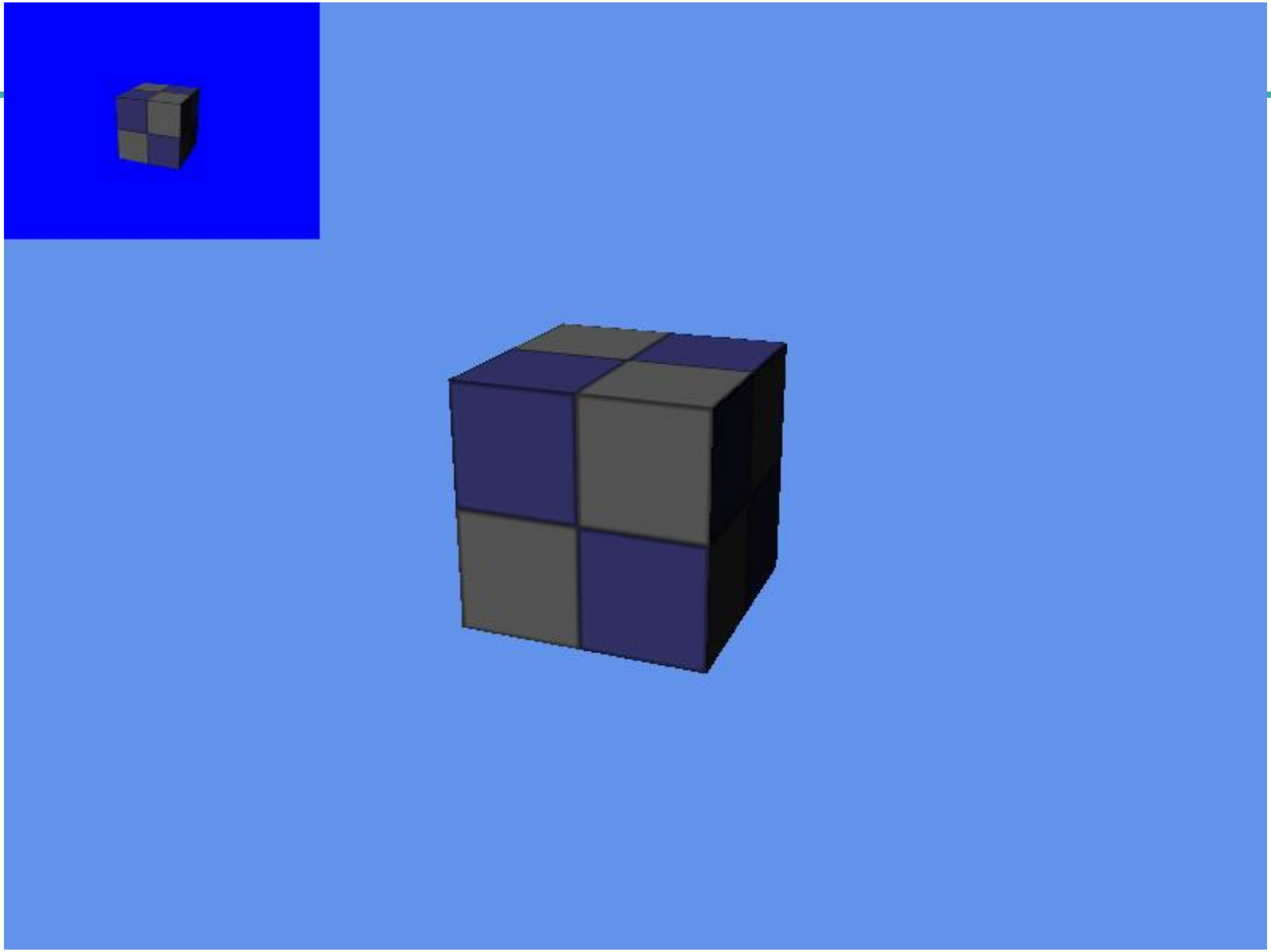
```
// To render ortho mesh
// Turn off the Z buffer to begin all 2D rendering.
Renderer->setZBuffer(false);

// ortho matrix for 2D rendering
orthoMatrix = Renderer->getOrthoMatrix();
orthoViewMatirx = camera->getBaseViewMatrix();

orthoMesh->sendData(renderer->getDeviceContext());
textureShader->setShaderParameters(renderer->getDeviceContext(), worldMatrix, orthoViewMatrix, orthoMatrix,
renderTexture->getShaderResourceView());
textureShader->render(renderer->getDeviceContext(), orthoMesh->getIndexCount());

renderer->setZBuffer(true);

// Present the rendered scene to the screen.
Renderer->endScene();
}
```



Warning

- You will need the render to texture working by following lab session
  - Once we have it working we are going to do something constructive with it
  - Most post processing requires multi pass rendering

Next week

- Week 7
    - Will be emailed more detailed info
    - No normal class
    - Special task
- Module survey?