The Application Stage

The Game Loop, Resource Management and Renderer Design

Application Stage Responsibilities

- Set up the rendering pipeline
- Resource Management
  - 3D meshes
  - Textures
  - etc.
- Prepare data for rendering (primitives, textures, etc.)
  - Collision detection
  - Frustum Culling
- Sends data down the pipeline and modifies pipeline state
  - Draw Calls
  - Changing shaders, textures, rasterizer states, blend states etc.

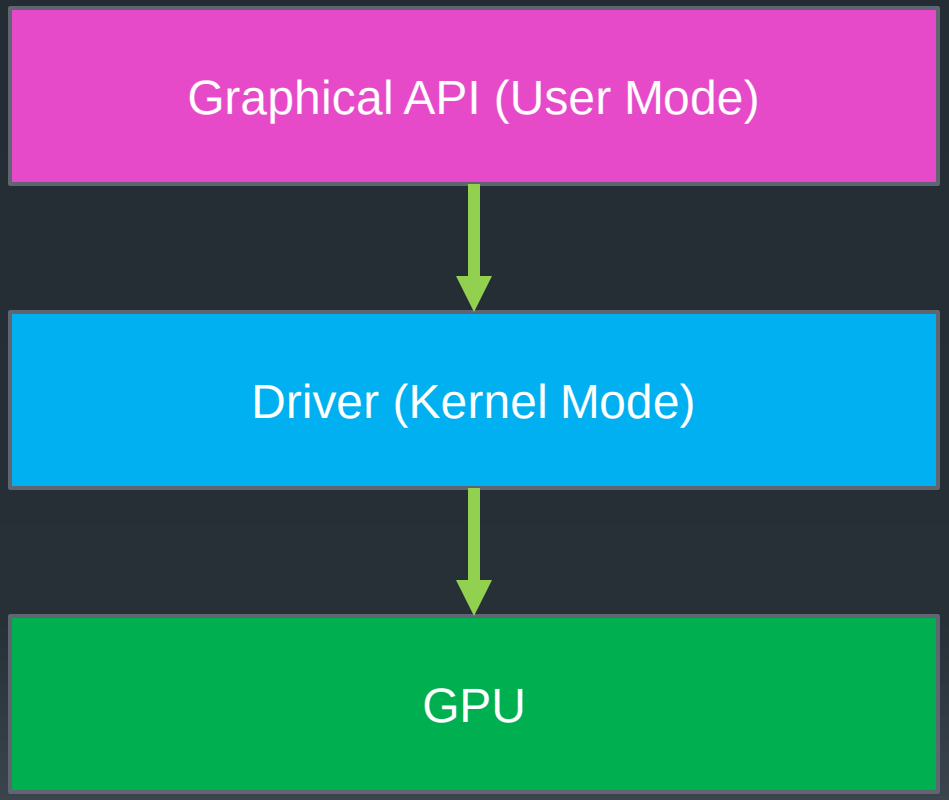
Virtual GPU Device Creation and Setup

The Graphics API and the Renderer

The Graphics API

- We use a graphics API to set up the GPU pipeline and to send data down the pipeline.
- A graphics API is low-level library that communicates with the GPU driver and allows us easy access to GPU features.
- The most common APIs are openGL and Direct3D.

The Graphics API



- The Graphical API sends commands to the Kernel Mode Driver.
- The driver then sends the actual commands to the GPU.
- There is a level of overhead introduced due to both the driver and the API.

What are the benefits of an API?

The Renderer and the Application

- MVC Design Pattern
- The Renderer – The View
  - Responsible for rendering the scene
  - Written in a graphics API
  - Handles all rendering and rendering resource management
- The Application – The Model
  - Determines what needs to be drawn
  - Runs all object updates (physics, collision detection etc.)
- User Input Handler – The controller

The Game Loop

- While (!exit)

- **Process User Input**
  - Keyboard/mouse Input
  - GUI Input

Controller

- **Update Game**
  - AI
  - Physics
  - Collision Detection
  - GUI updates

Model

- Render Scene
  - 3D Scene
  - GUI

View

Vertex Buffers, Index Buffers, Textures, Shader Constants

GPU Resource Management

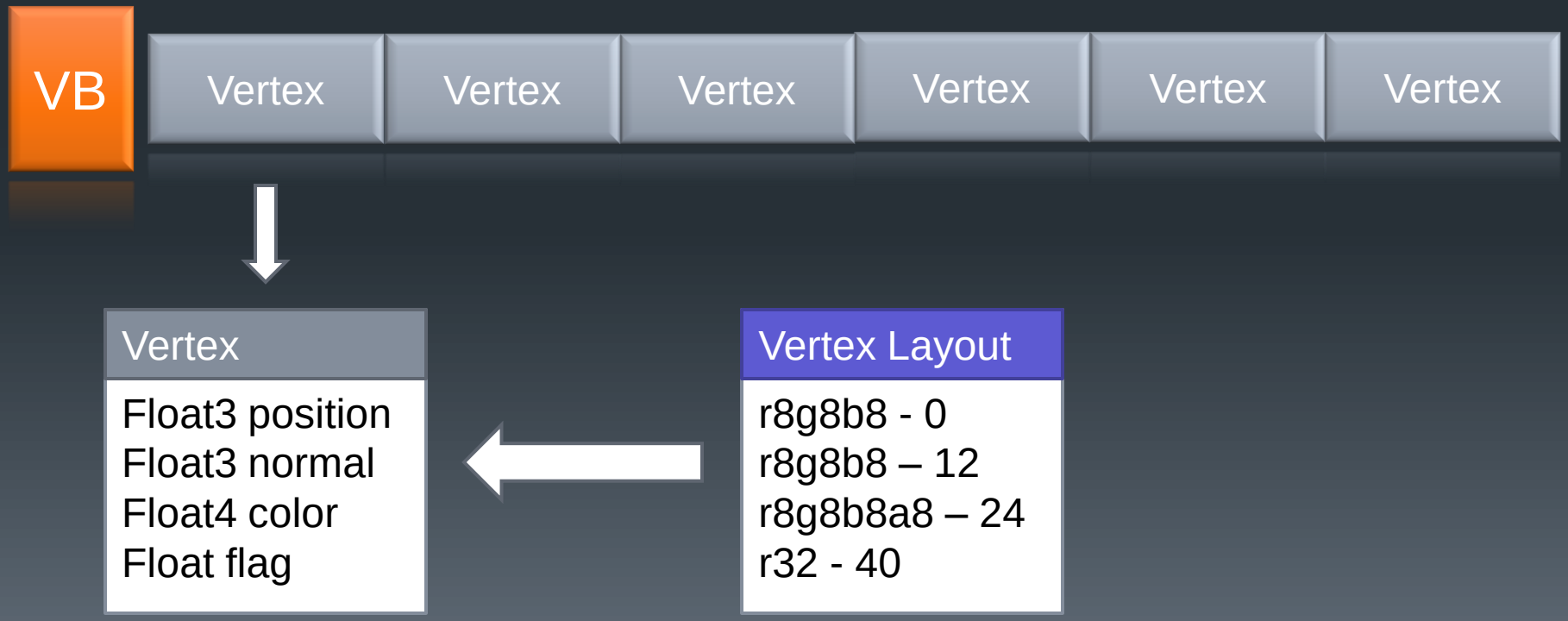
Resource Types

The most common GPU resources are:

- VERTEX BUFFERS
- INDEX BUFFERS
- TEXTURES

Vertex Buffers

- Large arrays containing vertex data
- Stored in video memory
- Requires input layout to describe memory layout of each element in the array

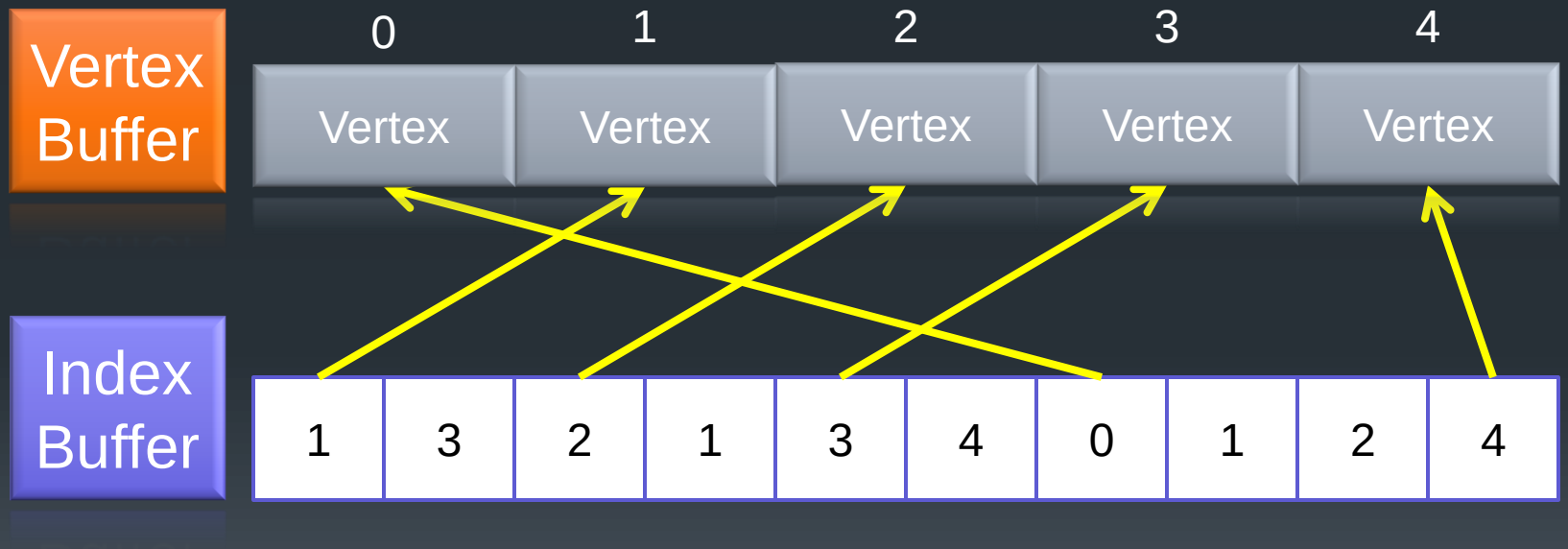


Vertex Buffers

- Static Buffers
  - Initialized with vertex data when created
  - Cannot be changed via CPU
  - Useful for static meshes
- Dynamic Buffers
  - Empty memory when created
  - Can be accessed and modified by CPU
  - CPU access performed via Mapping
  - Useful for changing vertex data, i.e. particle systems, text

Index Buffers

- Contains a list of vertex buffer indices
- Stored in video memory

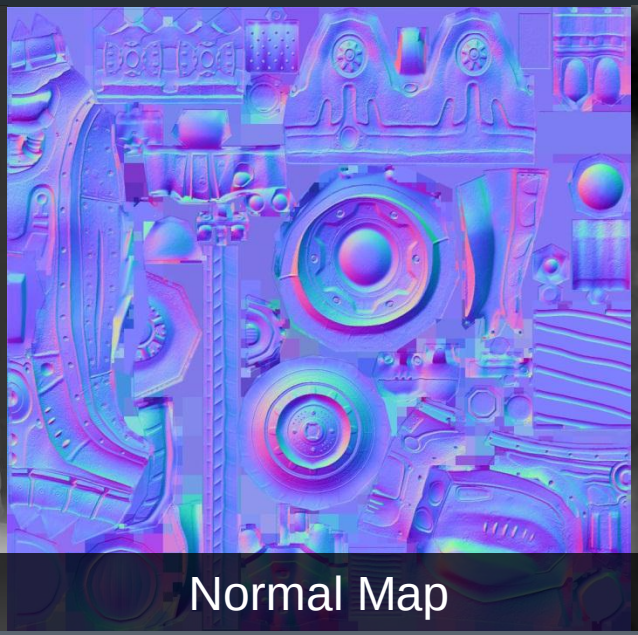


Why is this useful?

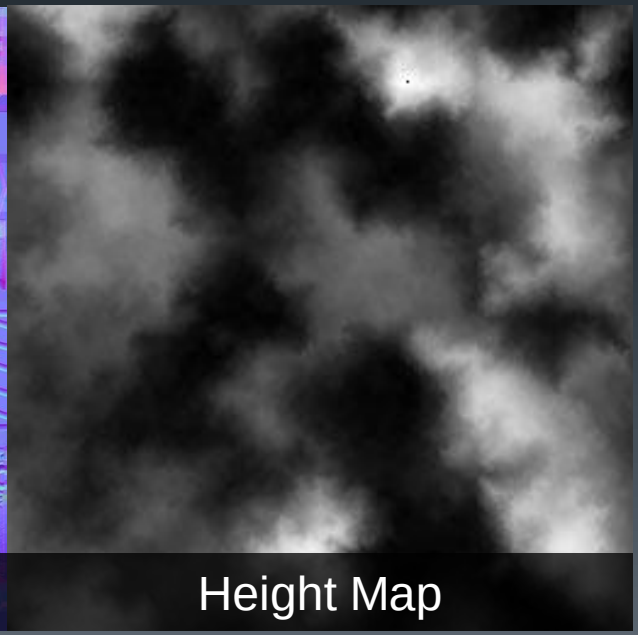
Textures

- Large arrays of data values stored in video memory
  - 1D,2D,3D
- Textures don’t necessarily contain image data
  - Height Maps, Normal Maps, Depth textures

Depth Texture



Normal Map



Height Map

Resource Management

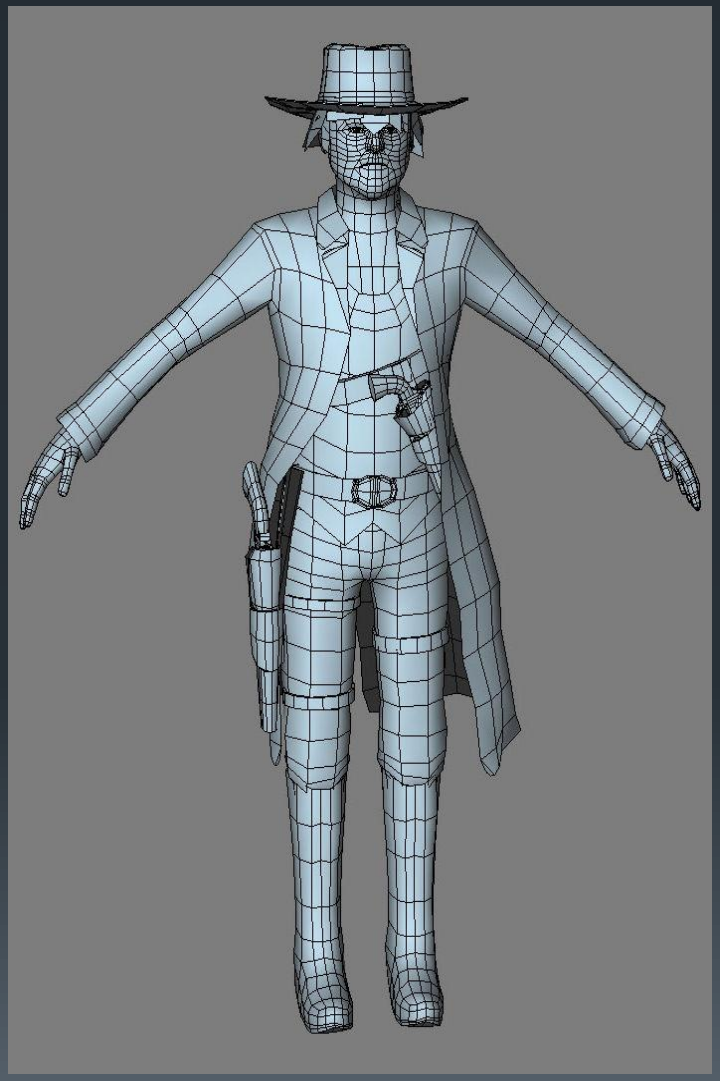
- Limited Video Memory
- Remember to “RELEASE” resources when finished with them – do not just delete the pointer!

- Low-level GPU resource management is the job of the renderer
- Simulation/Game will determine what resources need to be loaded or freed.

A trip down the pipeline

3D Rendering Basics

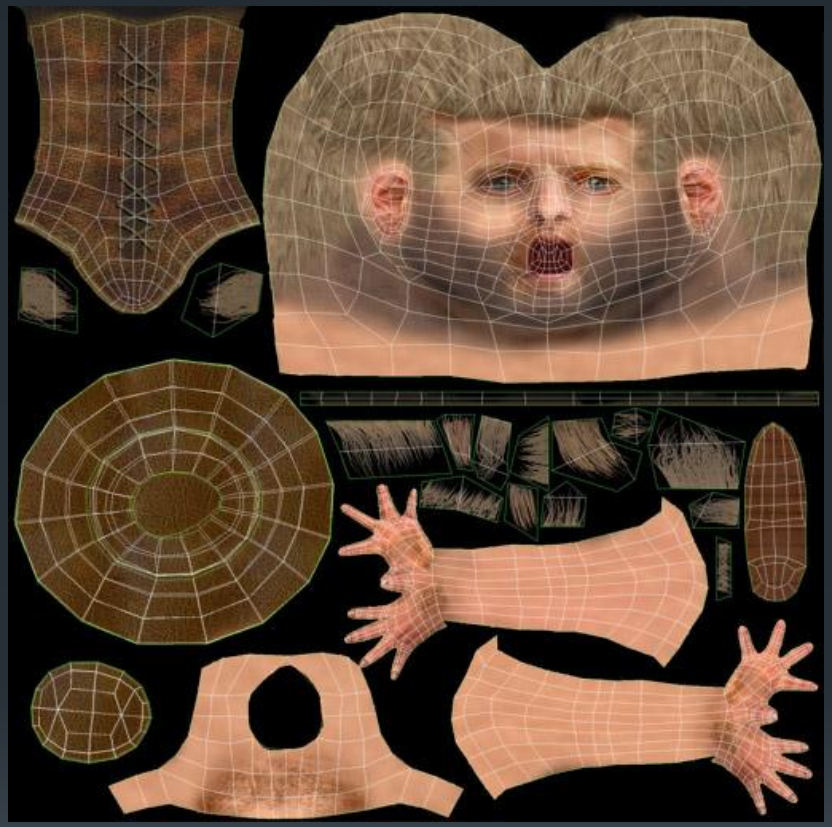
Rendering Basics – 3D Meshes



Model and art courtesy of quigmire.com

- All 3D objects are comprised of a set of connected convex polygons (faces). Each face is defined by a set of points (vertices).
- A model is often defined as a list of vertices, with additional information on how to connect the vertices to create the required faces.
- Each vertex is positioned relative to the models own co-ordinate system (space), the objects co-ordinate system is known as model space
- If needed texture mapping and lighting information is defined at each vertex.

Rendering Basics - Texturing



Model and art courtesy of quigmire.com

- To add more detail to a model we “skin” or wrap the model using a 2D image which defines how the surface of the object should appear.
- Each face is mapped to portion of the 2D image by a technique called texture mapping or UV mapping.
- *UV mapping refers to the co-ordinate system used by the texture. The texture is 2D and u represents the horizontal axis and v represents the vertical axis.*
- Mapping is the process of representing one coordinate system in terms of another.

Rendering Basics - Lighting

Model and art courtesy of quigmire.com

- The previous mesh had very little detail in it, and so the artist has improved it by increasing the polygon (face) count. This is called a high polygon model. High polygon models are smoother in appearance and allow for more detailed lighting.
- Additional Lighting parameters are now added to the model so that it can be correctly shaded.
- In the model we can see the illusion of details like button holes and stitching, these illusions are achieved by advanced lighting techniques such as normal mapping.

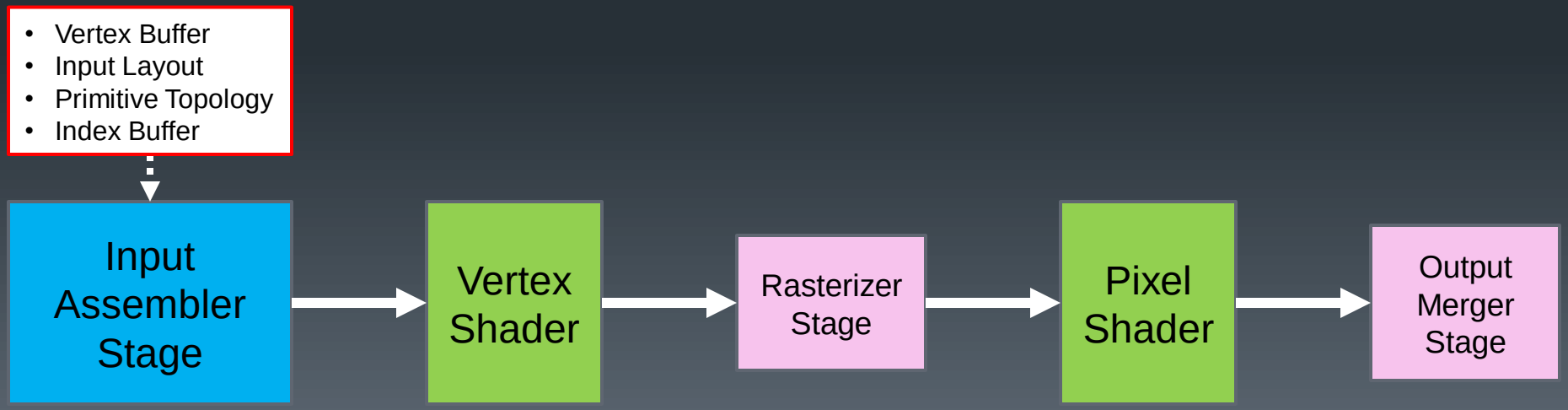
Rendering Basics – Final Result

Model and art courtesy of quigmire.com

- The image shown here is the result of the combination of the three basic techniques we just described.
- All the information needed to render the model was defined in the application stage, the vertex positions relative to the model's coordinate system, all the necessary resources such as textures and bump maps, all necessary lighting info such as materials and normal vectors are set up prior to sending anything to the GPU.
- Once all the data and resources have been allocated for the model then we send all the data down the pipeline using "draw calls".

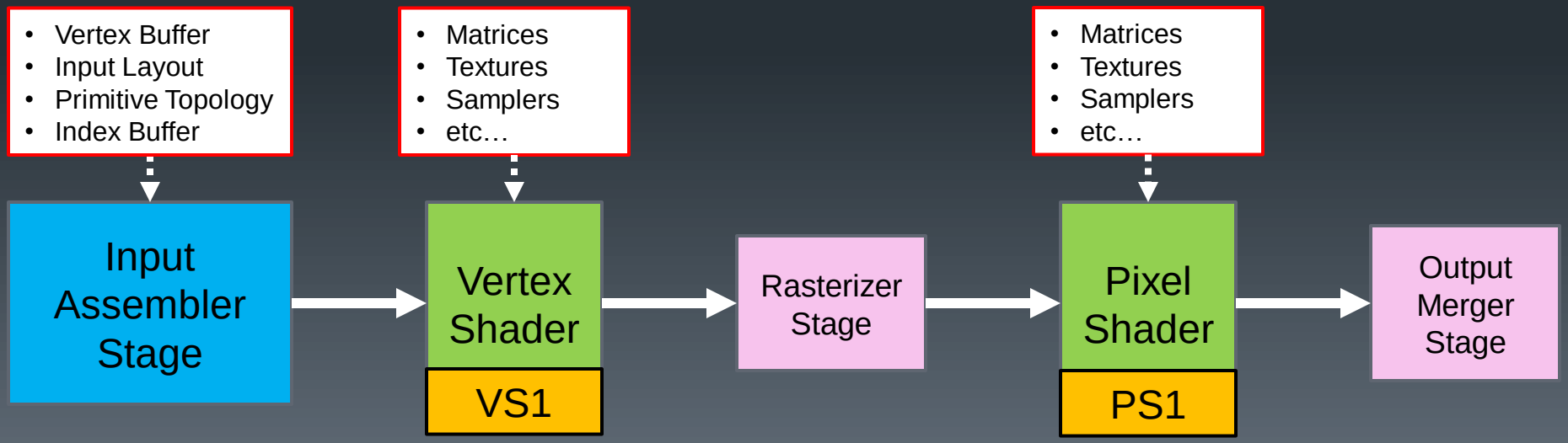
A trip down the Pipeline

- Okay so how do we send a 3D object down the pipeline?
- First step is to set up the input assembler stage:
  - Bind the vertex buffer containing the object to the IA stage
  - Set the input layout of the vertex buffer
  - Set the primitive topology of the object's vertex buffer
  - Bind any index buffers used to draw the object



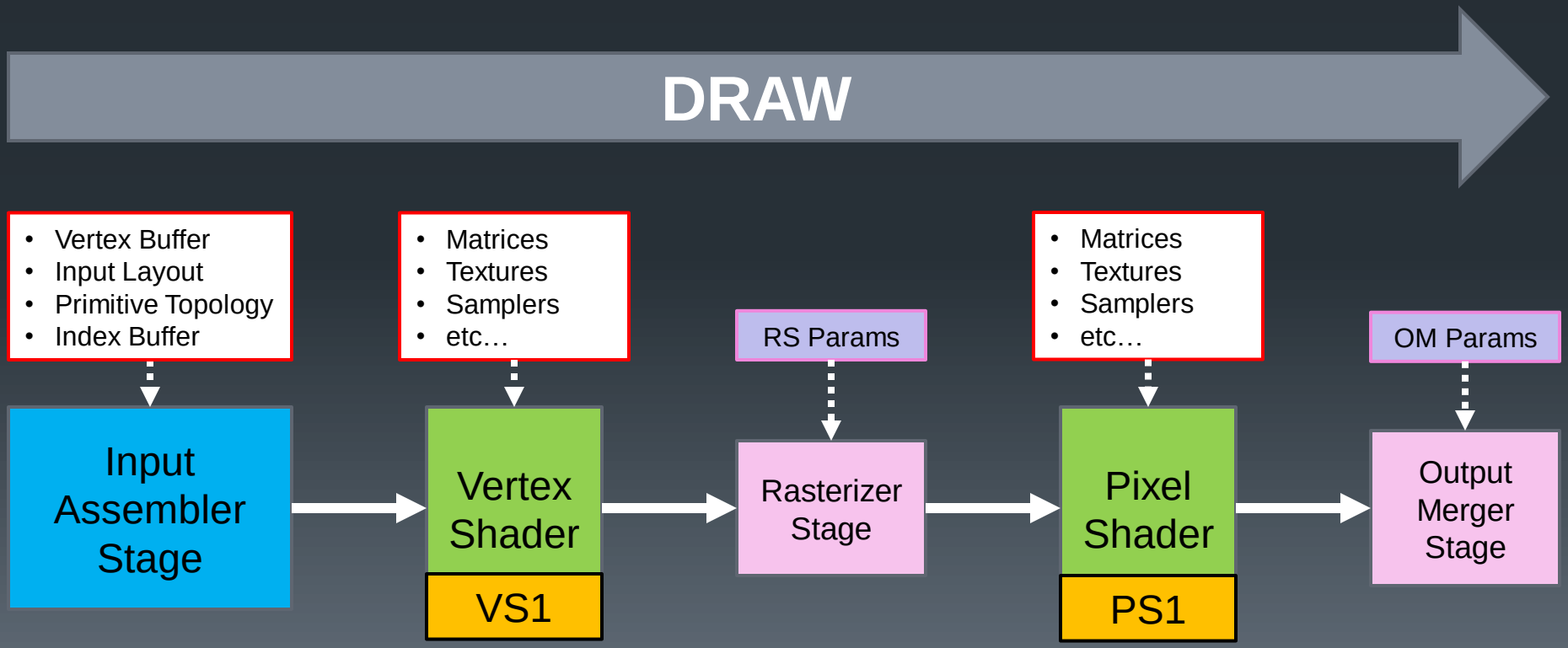
A trip down the Pipeline

- Now we need to set the vertex and pixel shader programs.
- Then we need to set up all vertex shader data
  - matrices, constants, textures, samplers, etc…
- Then we need to set up all pixel shader data
  - matrices, constants, textures, samplers, etc…



A trip down the Pipeline

- The final step is to configure the Rasterizer and Merger stages.
- Now we can call the DRAW function to send all the data down the pipeline.



Direct3D 10 Effect System

- The Direct3D 10 Effects contains the entire pipeline state apart from the IA stage.
- All stage data (matrices, textures, etc.) need to be set via the effect interface.
- When you apply a technique you set the pipeline state (including all stage data).