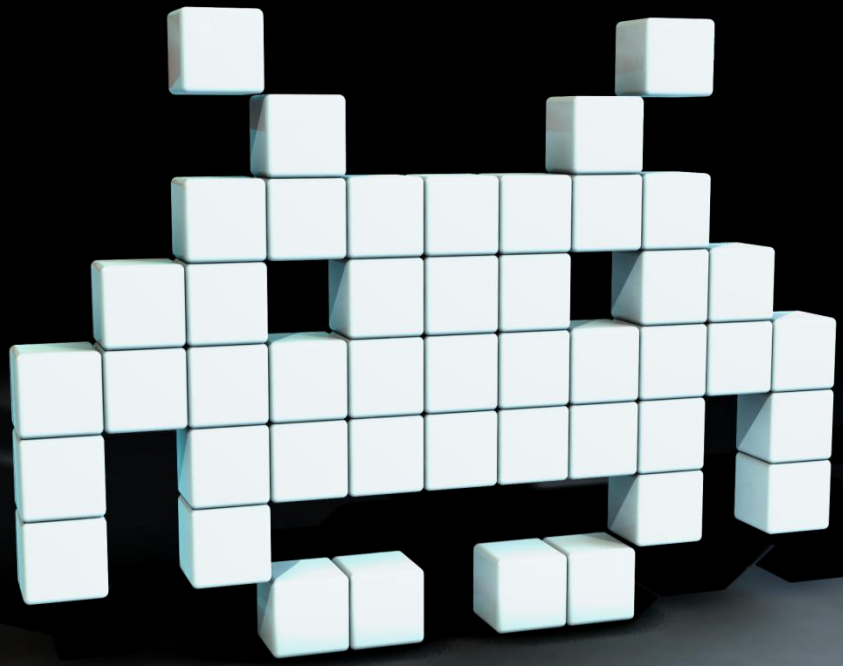


Summer Games University

Day 3: Graphics

Now in color and realtime

Image sources: Vergemce, Crytek

Connecting the Engine

Let's take a look on the interface an engine might have and find some good patterns on how to model them.

Read-only

Create a manager instance in the engine, which translates incoming data (Player Input, ..)

Write-only

Use a renderer to create a representation of required information (Graphics, Sound, ..)

Communication

No generic pattern. You will have to integrate it into the architecture (Physics, ..)

Connecting the Graphics

First idea:

~~Every gameobject has a Render() method~~

This is your architecture then

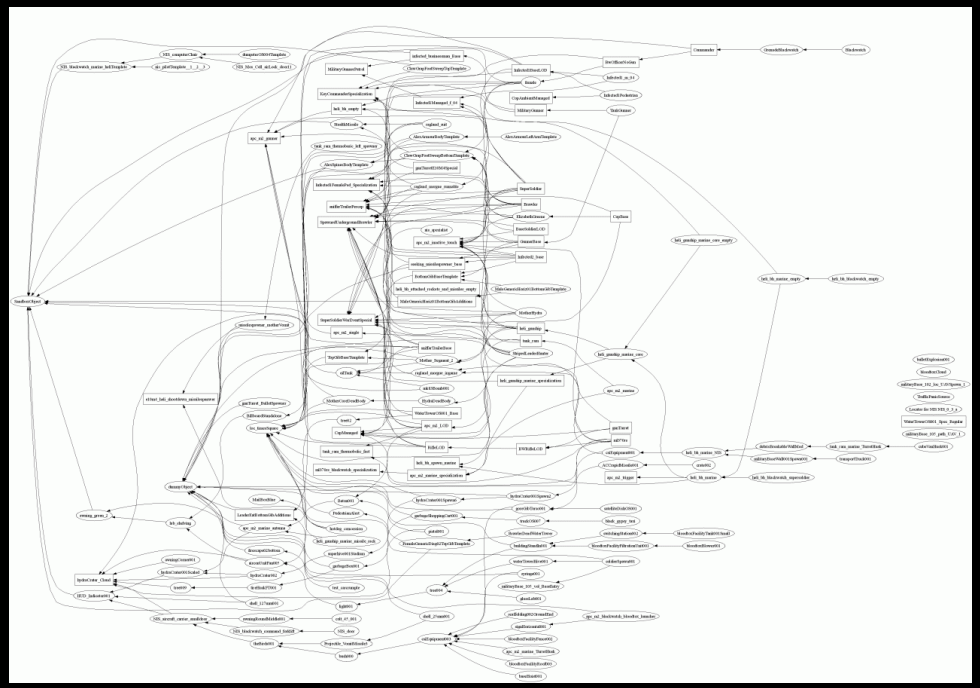
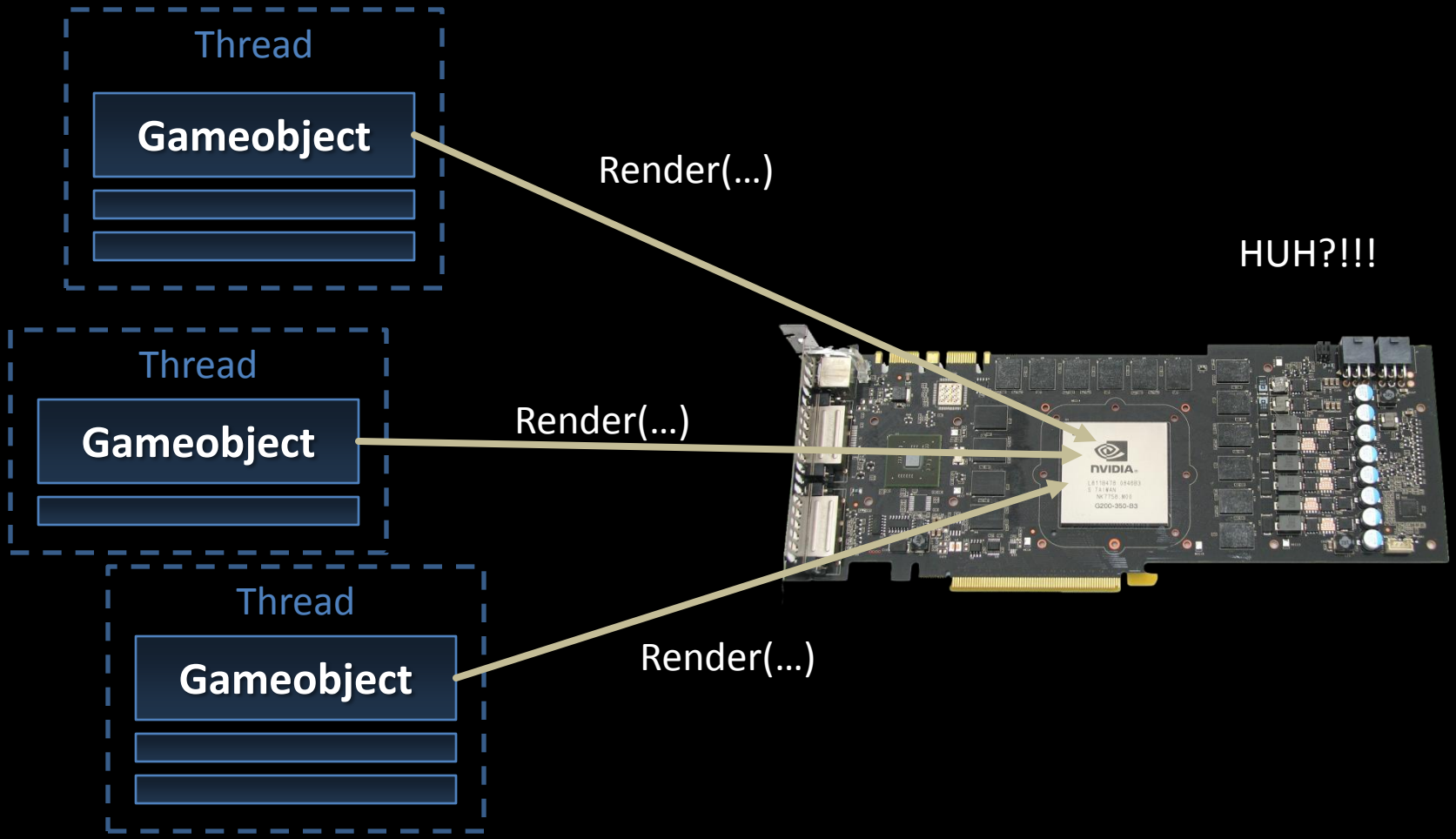


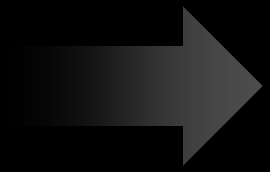
Image source: marcinchady GDC Canada Behavior presentation

Multithreading



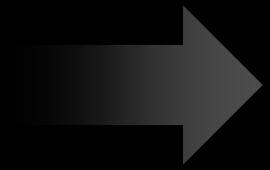
Multithreading

Rendering is a pipelined process



Only one Renderer

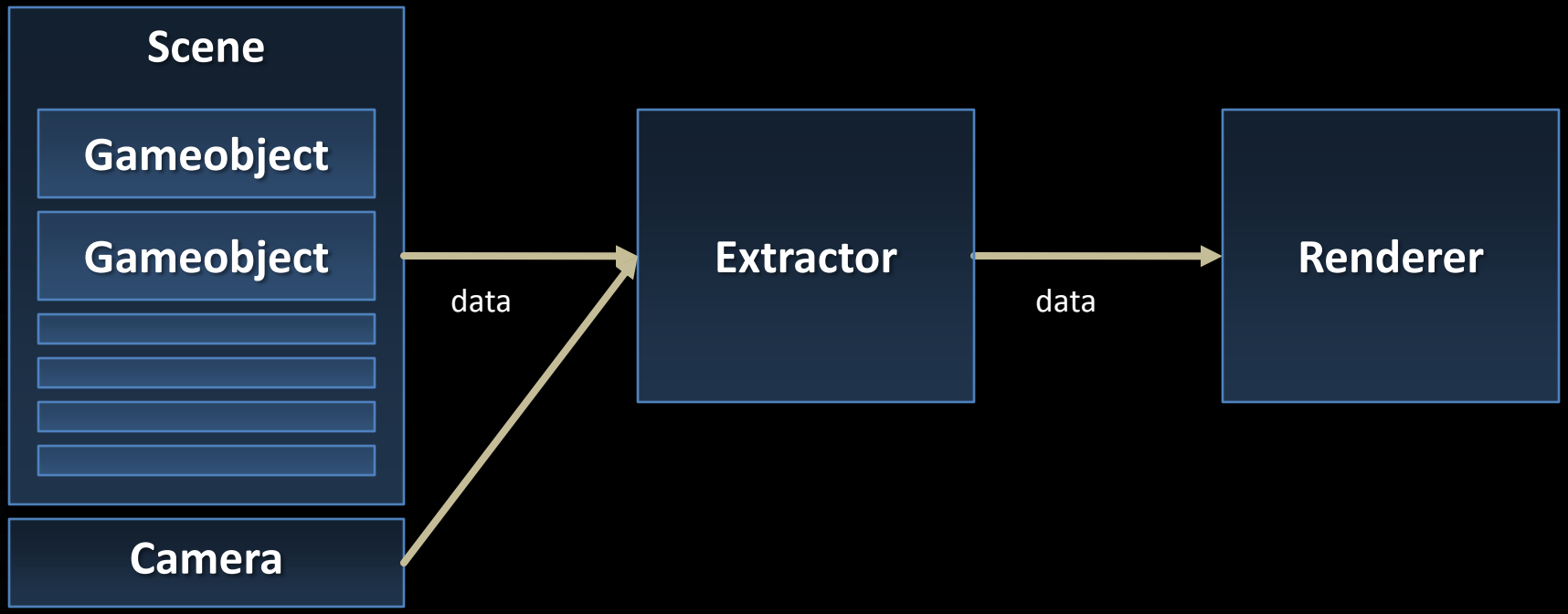
Random order results in many state changes



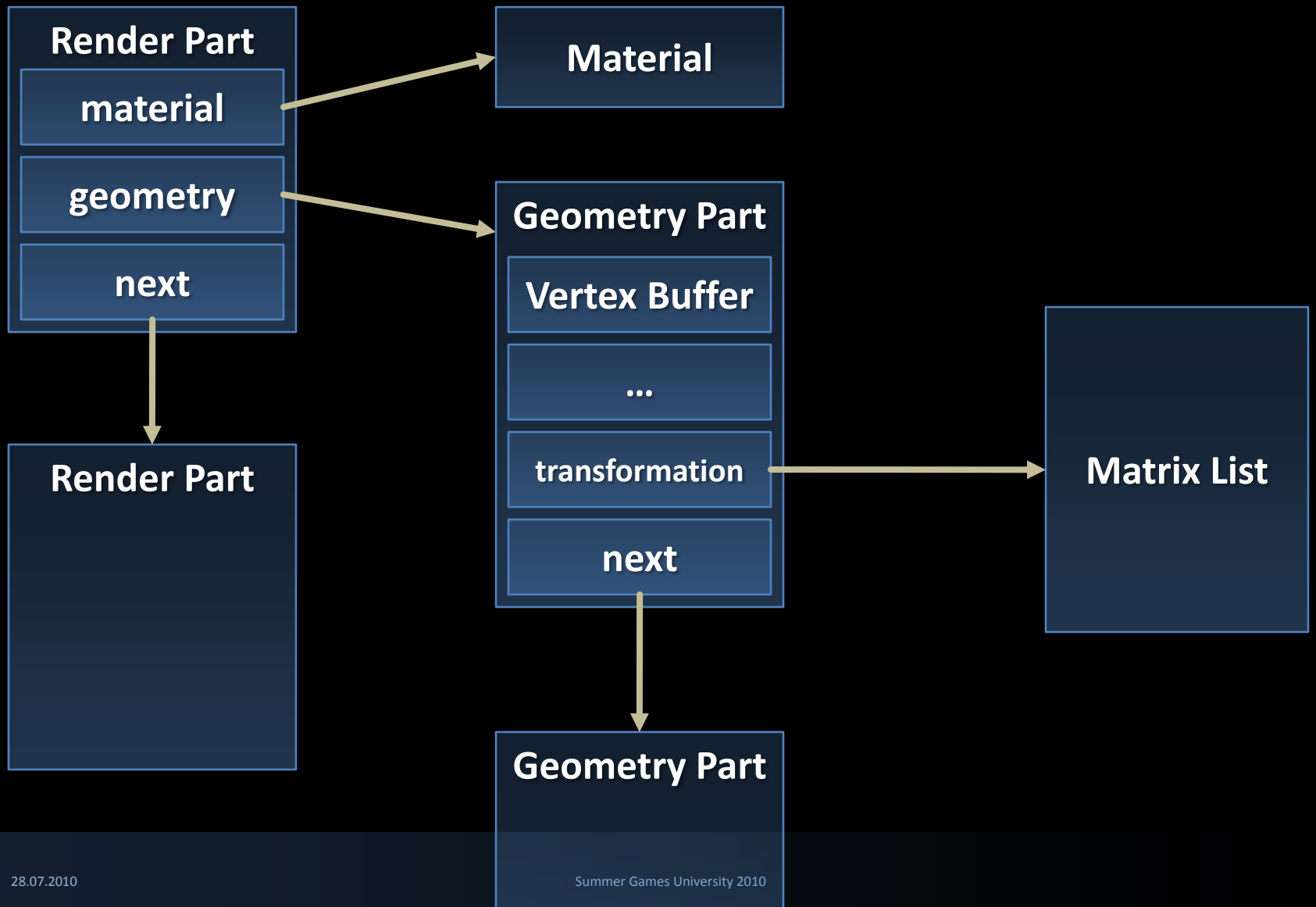
Defined Order

Connecting the Graphics

Better: data driven architecture

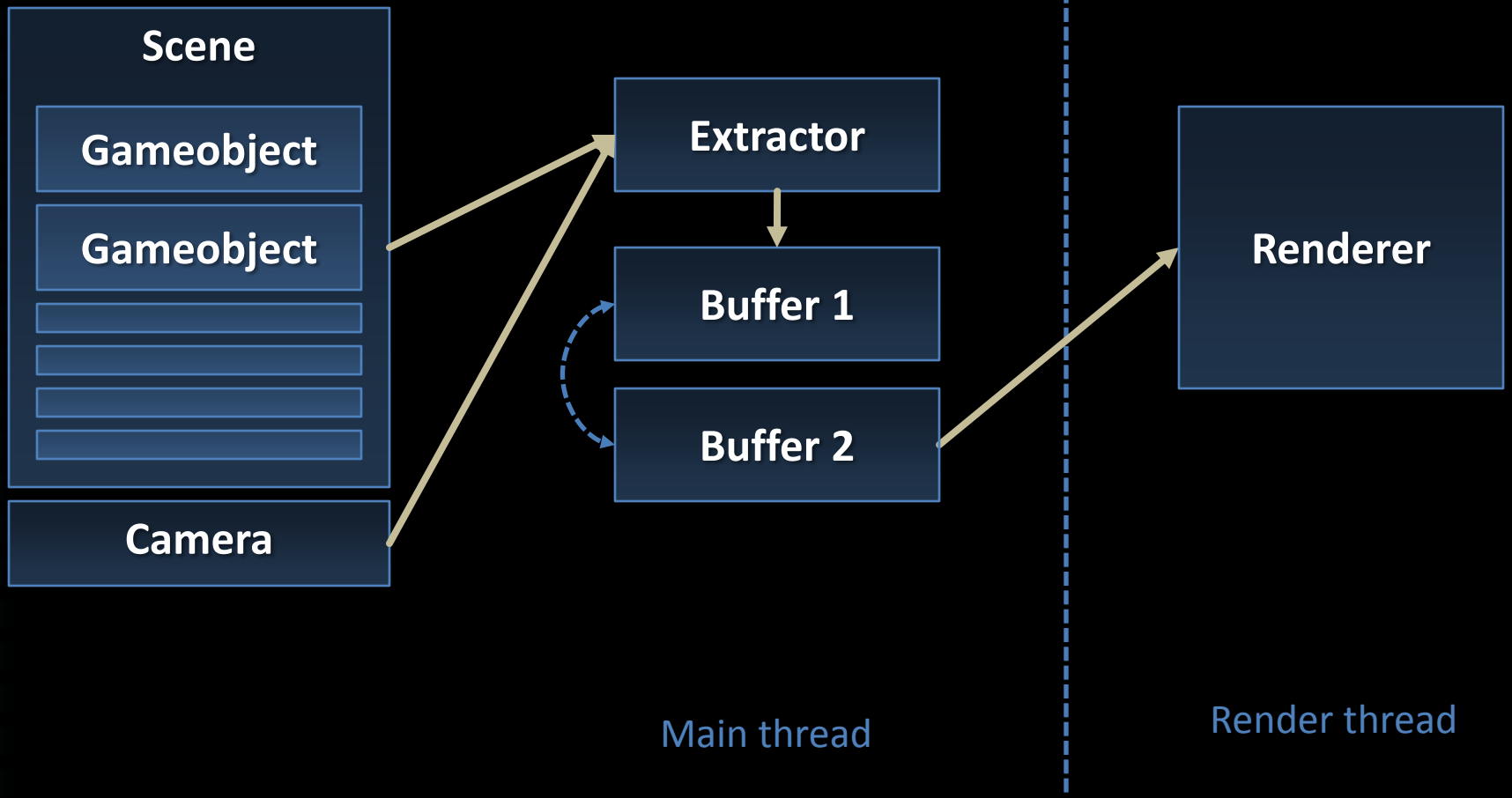


Connecting the Graphics



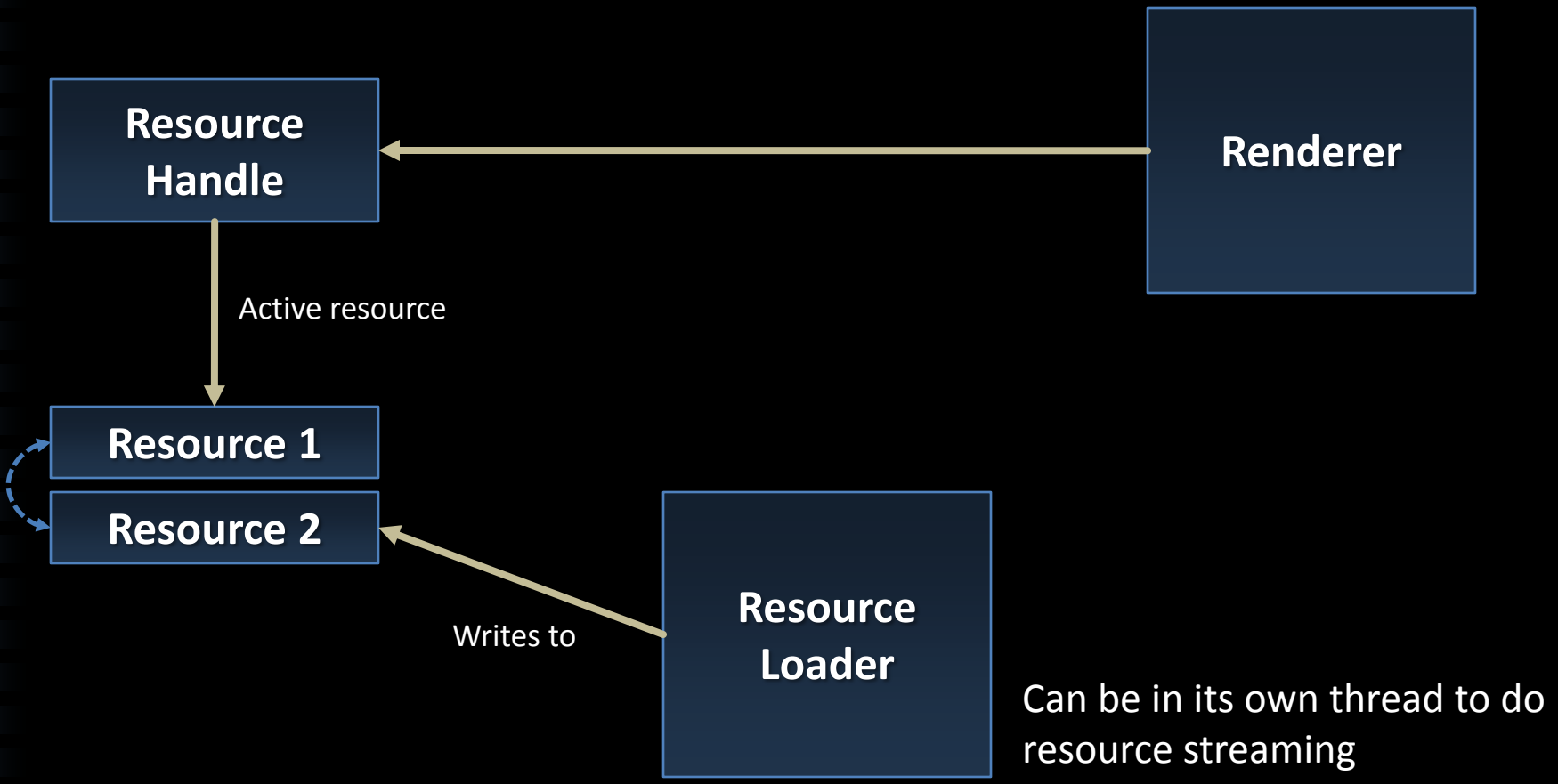
Connecting the Graphics

Multithreaded version



Connecting the Graphics

Resource Handles: Make them double buffered as well!

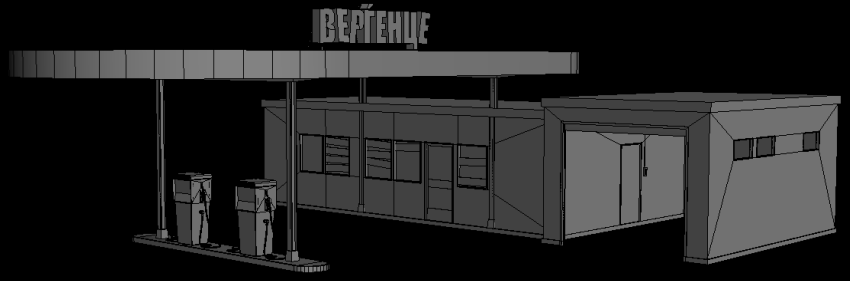


Can be in its own thread to do resource streaming

Let’s leave the architecture for a while…

Image source: Mumbai, Elphinstone Mills Tower, skyscraperpage.com

…let’s do some graphics



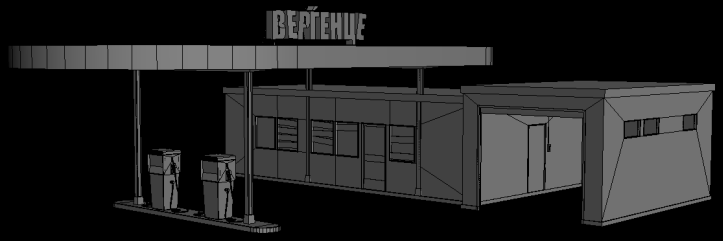
Geometry

Light

Material

X?

Geometry



3d modeling tool

Engine

Lighting Overview

Static → Dynamic

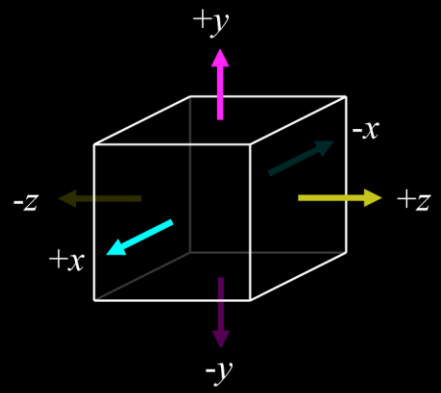
Image sources (left to right): “Half Life 2” Valve, “Unreal Tournament 3” Epic Games, “GTA4” Rockstar, “Crysis 2” Crytek

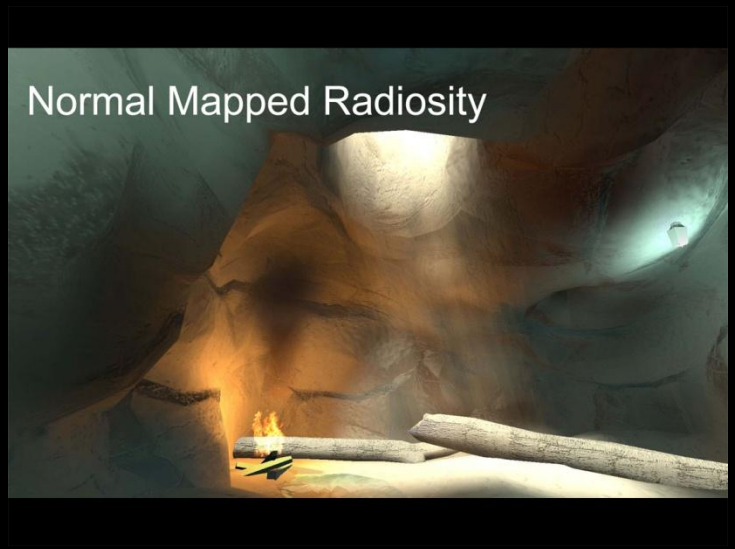
Static Lighting

Prerendered lightmaps stored in textures

Static Lighting

What about normalmaps and moving objects?





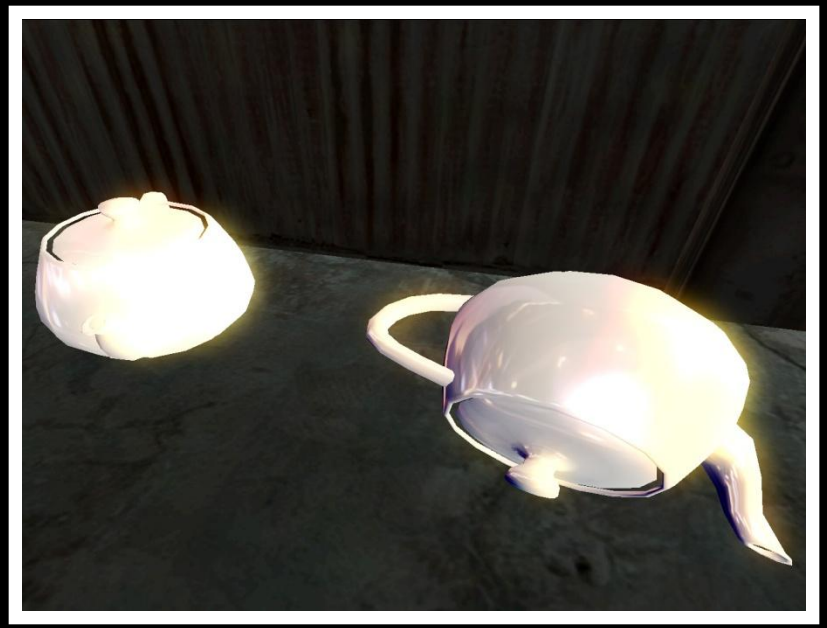
You need special tools to do this!

Source: “Half Life 2 Shading Paper” Valve (http://www.valvesoftware.com/publications/2004/GDC2004_Half-Life2_Shading.pdf)

Static Lighting

Simple solution for moving objects: Hemispherical ambient

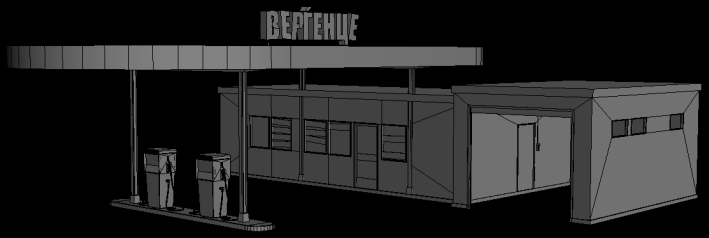
Looks great! (except for the missing shadows…)



Huh, radioactive teapots?

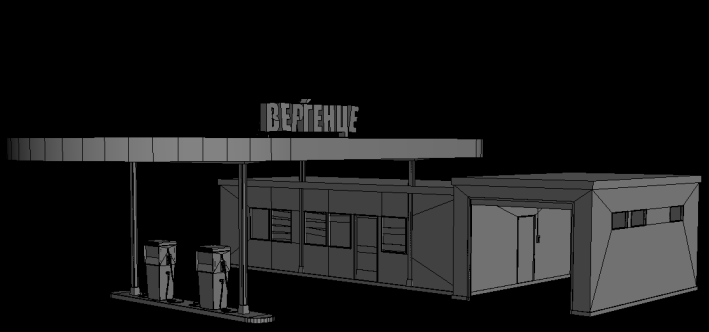
Dynamic Lighting

Forward Shading / Rendering

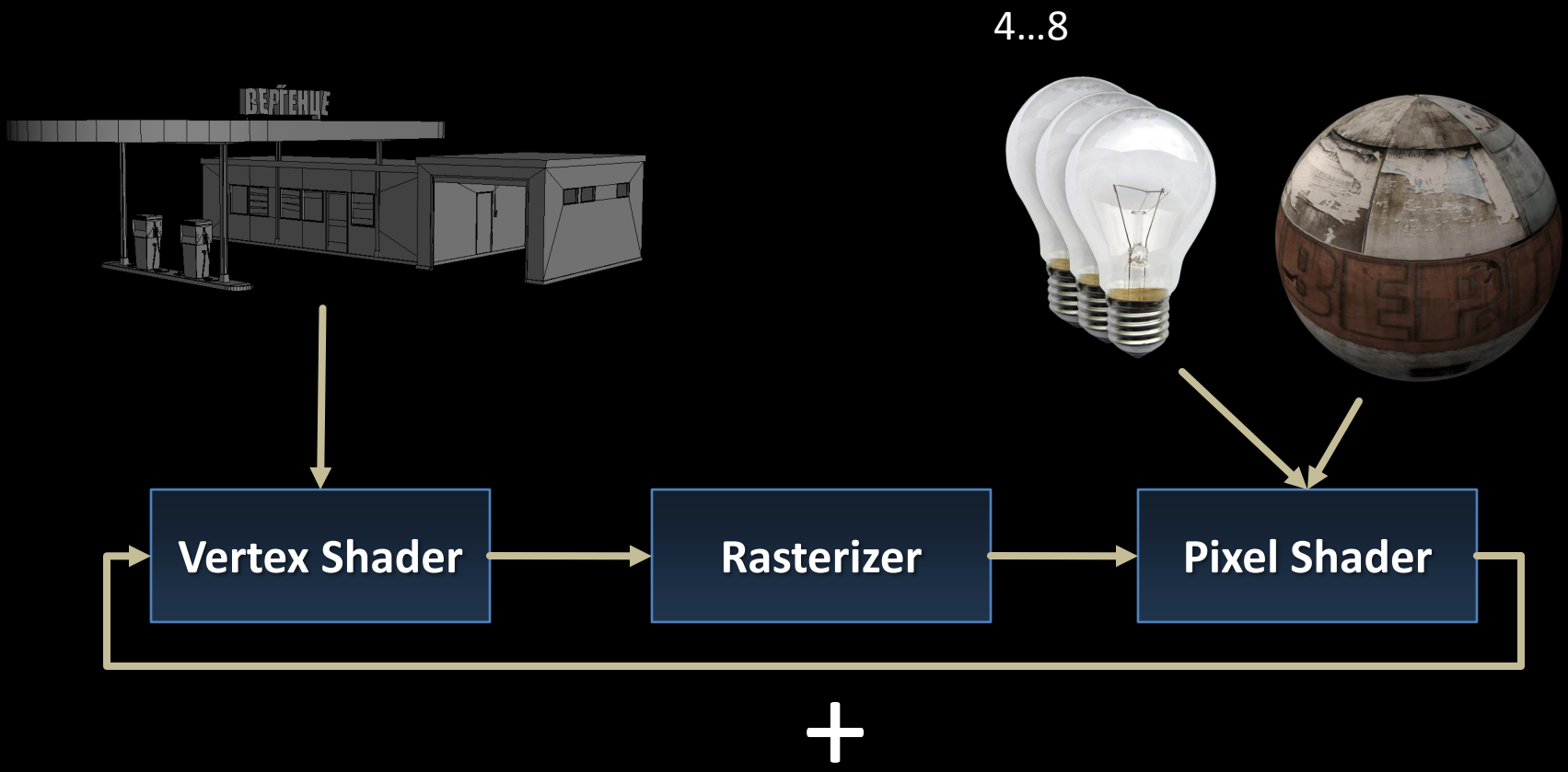


Forward Shading / Rendering

Problems?



Forward Shading / Rendering



Forward Shading / Rendering

Problems?

geometry overhead

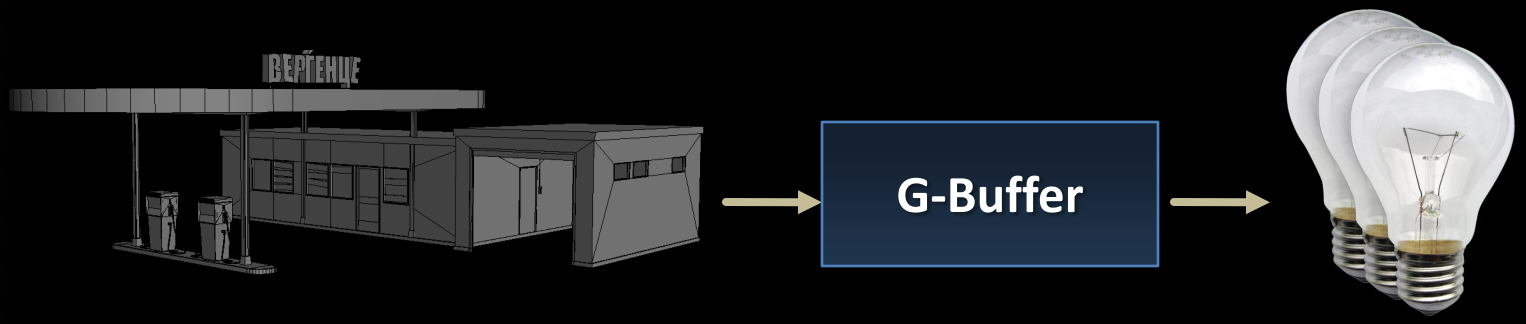
complexity O(G*L)

material-light combinations

unpredictable performance

Dynamic Lighting

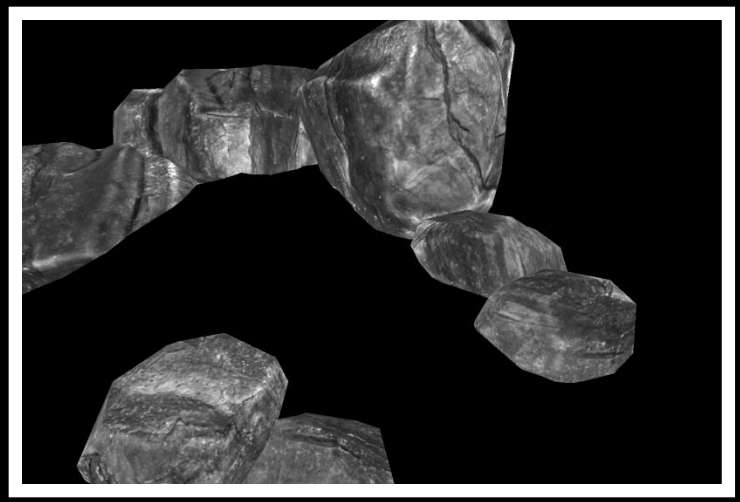
Deferred Shading / Rendering



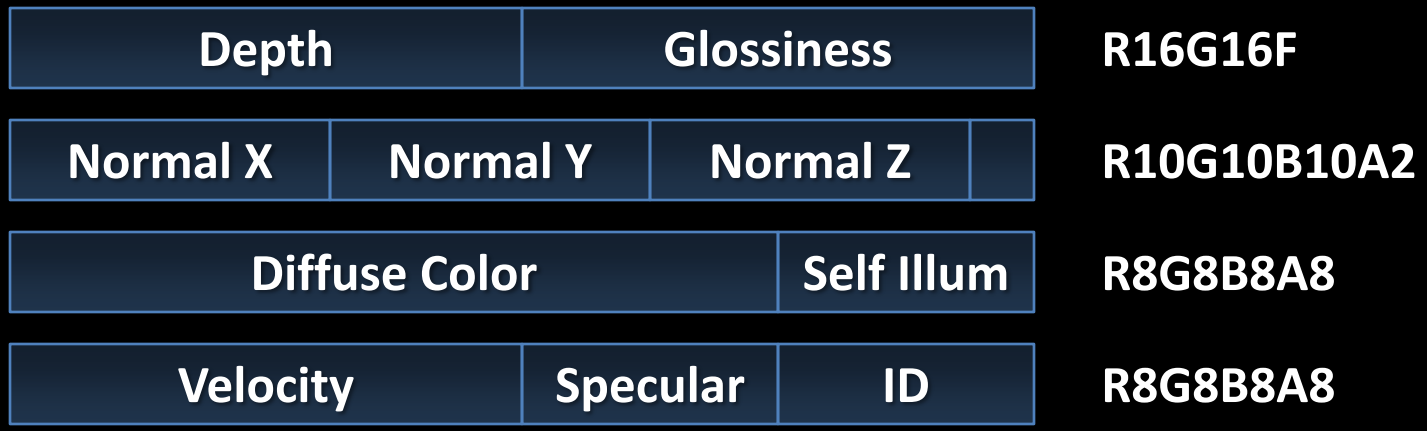
pass 1:
geometry

pass 2:
light

Deferred Shading / Rendering



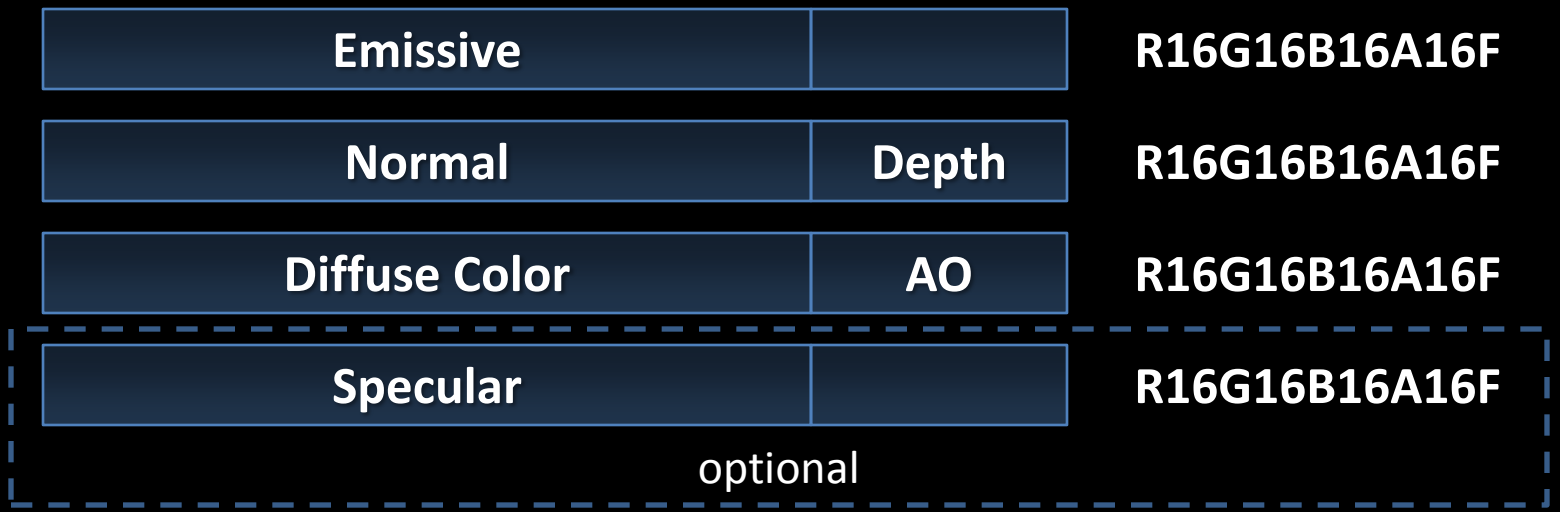
Deferred Shading / Rendering



*Depth and Normals are stored in Viewspace

Deferred Shading / Rendering

Starcraft 2



Source: StarCraftII Effects & Techniques

Deferred Shading / Rendering

Advantage

no geometry overhead

complexity O(G+L)

no need for “ubershader”

Deferred Shading / Rendering

Disadvantage

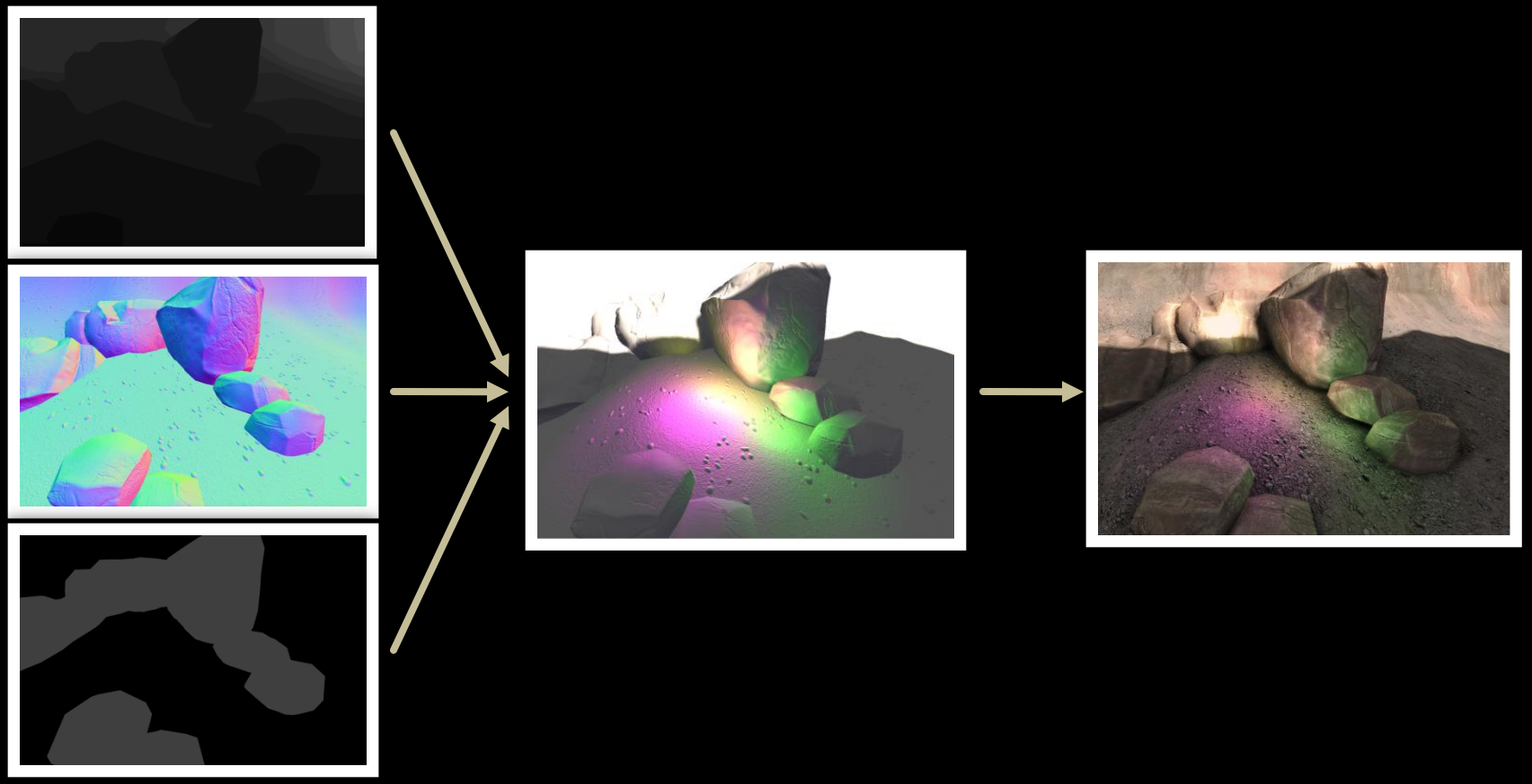
high memory bandwidth

no transparent geometry

no native anti-aliasing

restricted to one shading-type
(phong)

Light Prepass Rendering



pass 1:
geometry

pass 2:
light

pass 3:
material

Light Prepass Rendering

Advantage

- less memory bandwidth
- any possible shading type
- native anti-aliasing

Light Prepass Rendering

Disadvantage

render geometry twice

again no transparent geometry

Who uses it?

Forward Shading → Deferred Shading → Light Prepass

Image sources (left to right): “Doom 3” id Software, “Crysis” Crytek, “Stalker” GSC, “Starcraft 2” Blizzard, “GTA4” Rockstar, “Crysis 2” Crytek

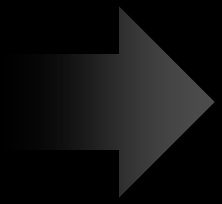
Going one step further...

Deferred

+ no geometry overhead
- high memory bandwidth
- no transparent geometry
- no anti-aliasing
- only one shading type

Light Prepass

- render geometry twice
+ less memory bandwidth
- no transparent geometry
+ anti-aliasing
+ any possible shading type



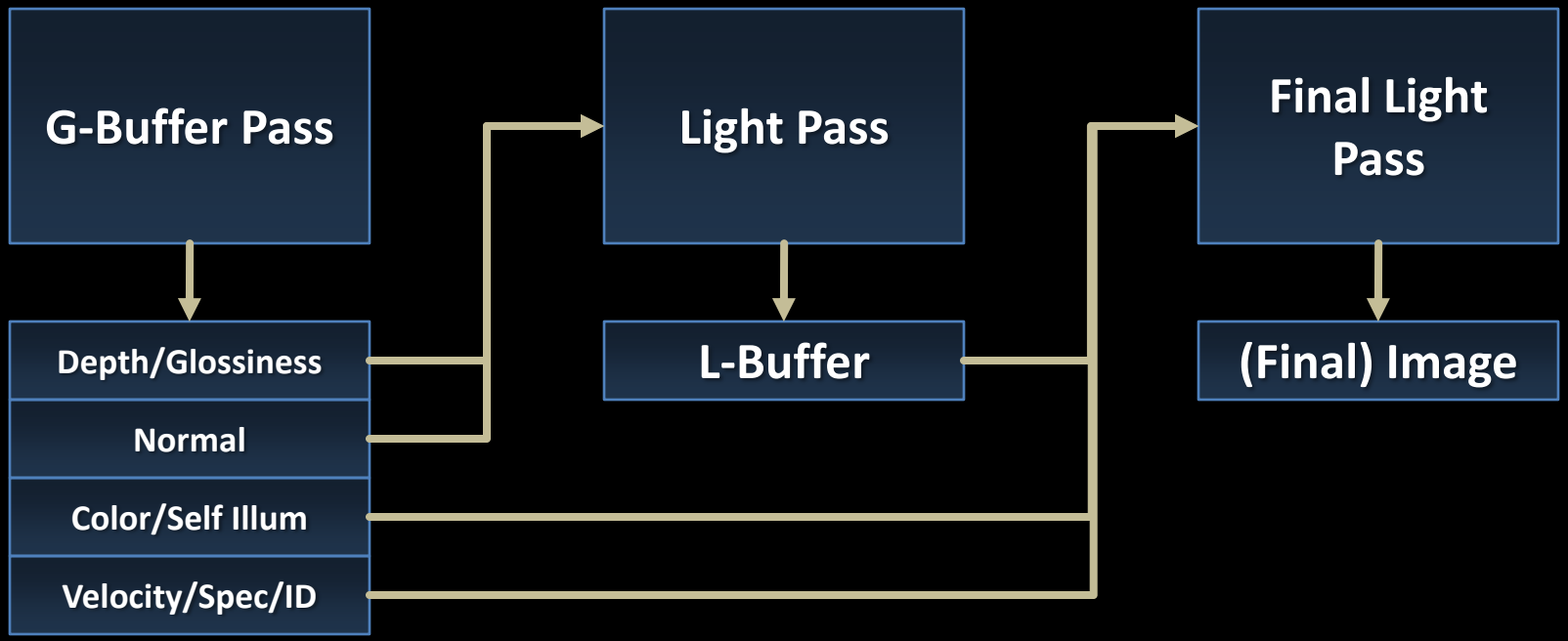
Take the advantages and mix both approaches together

Going one step further...

minimal G-Buffer as used for a light prepass renderer

Depth		Glossiness		R16G16
Normal X	Normal Y	Normal Z		R10G10B10A2
Diffuse Color			Self Illum	R8G8B8A8
Velocity		Specular	ID	R8G8B8A8

Renderpipeline



+ less memory bandwith
+ all objects using phong shading in one geometry pass
+ can do other shading types, because we have a Light Buffer (need to be renderer twice)
- anti-aliasing (can be done as post process)
- transparent geometry ☹

Demo

Transparent Geometry

Fallback to forward shading -> difficult to keep lighting consistent

Dithering

Dithering only in L-Buffer

Image sources: "GTA4" Rockstar, "**Inferred Lighting" Paper** Scott Kircher, Alan Lawrance

We still have no architecture for the renderer itself

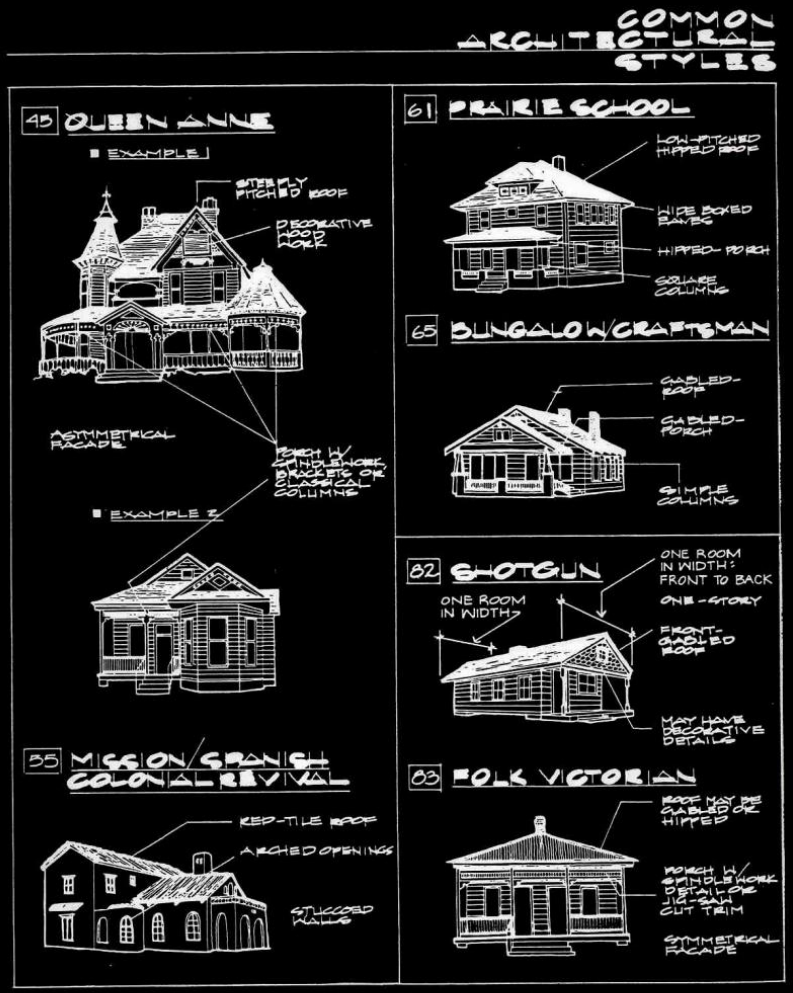
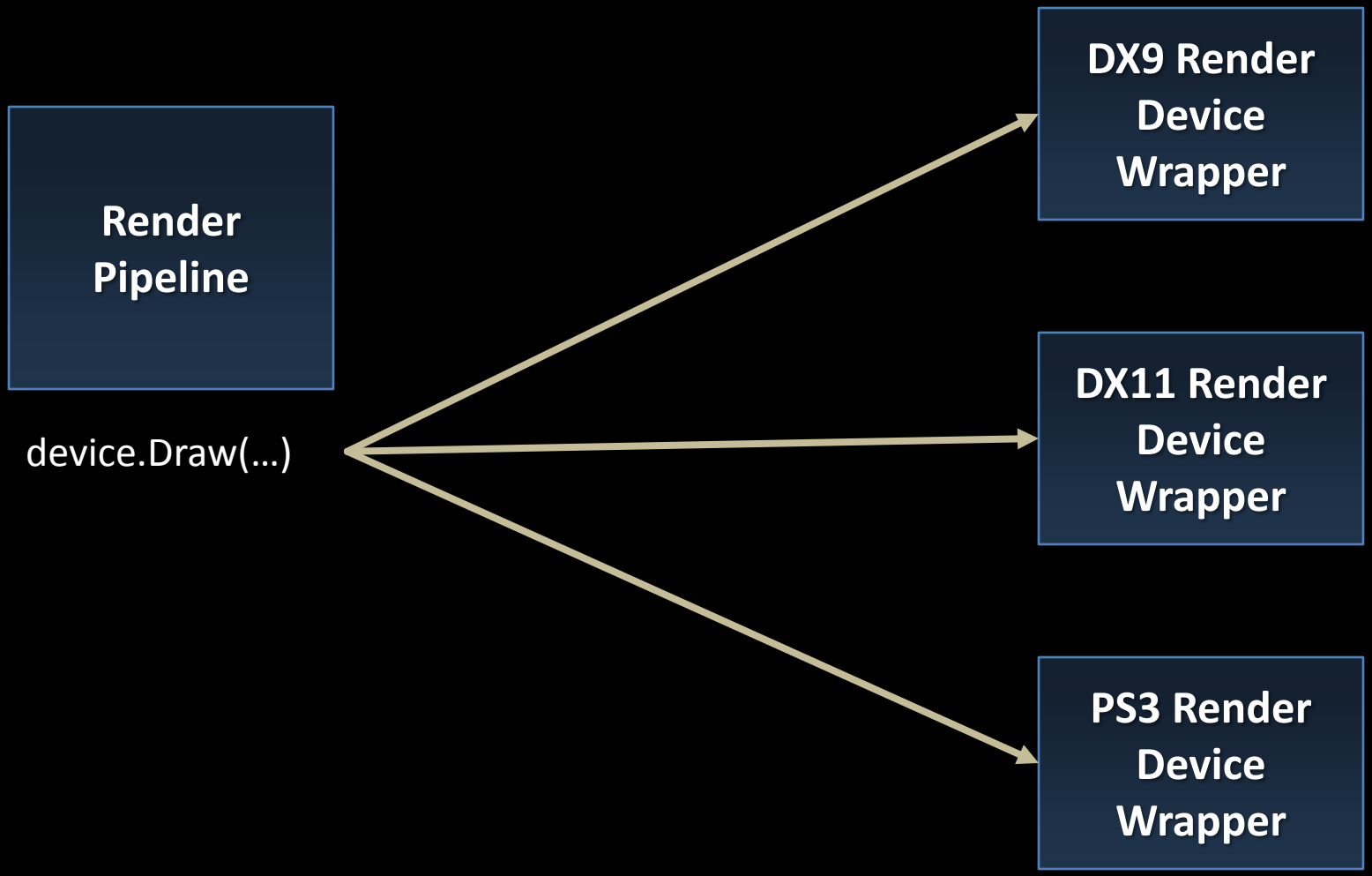
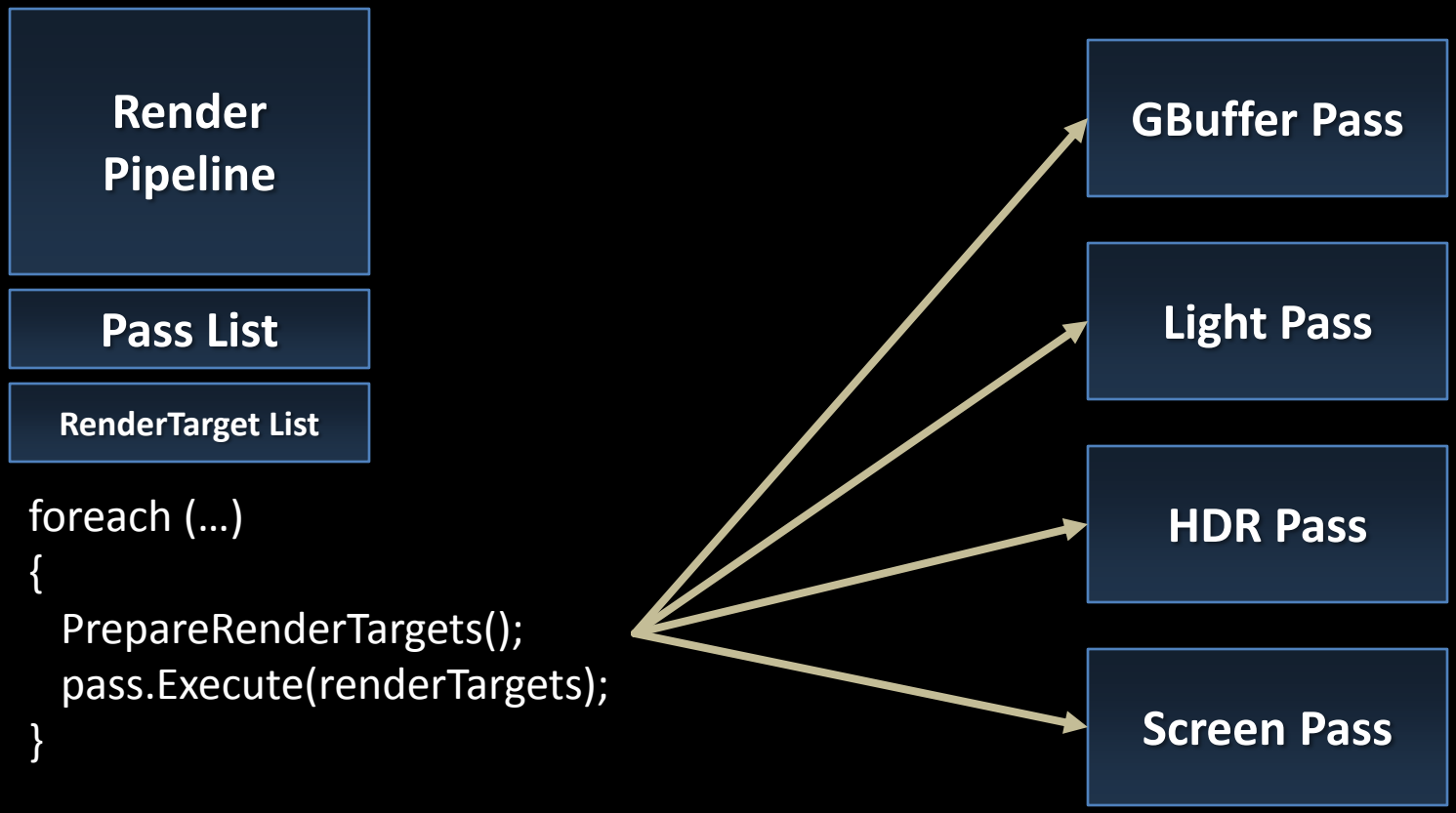


Image source: www.okhistory.org

Renderer Architecture

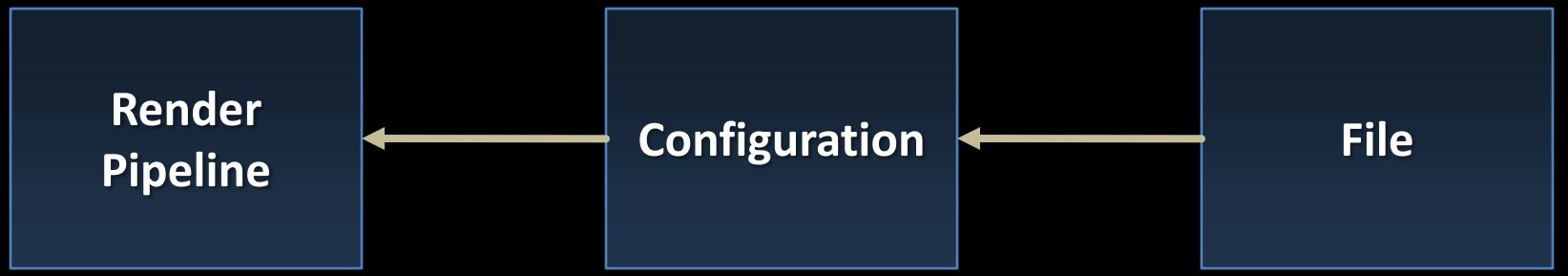


Renderer Architecture



Renderer Architecture

Data driven again:



Renderer Architecture

```
{
        rendertargets: [
                {rw:1, rh:1, f:10},
                {rw:1, rh:1, f:3},
                {rw:1, rh:1, f:1},
                {rw:1, rh:1, f:1},
                …
        ],
        passes: {
                GBufferPass: {
                        rt: [0, 1, 2, 3]
                },
                AmbientOcclusionPass: {
                        rt: [0, 4, 5, 6]
                },
                LightPass: {
                        rt: [0, 1, 2, 3, 6, 7, 8]
                },
                …
        }
}
```

Material

Shader

Material

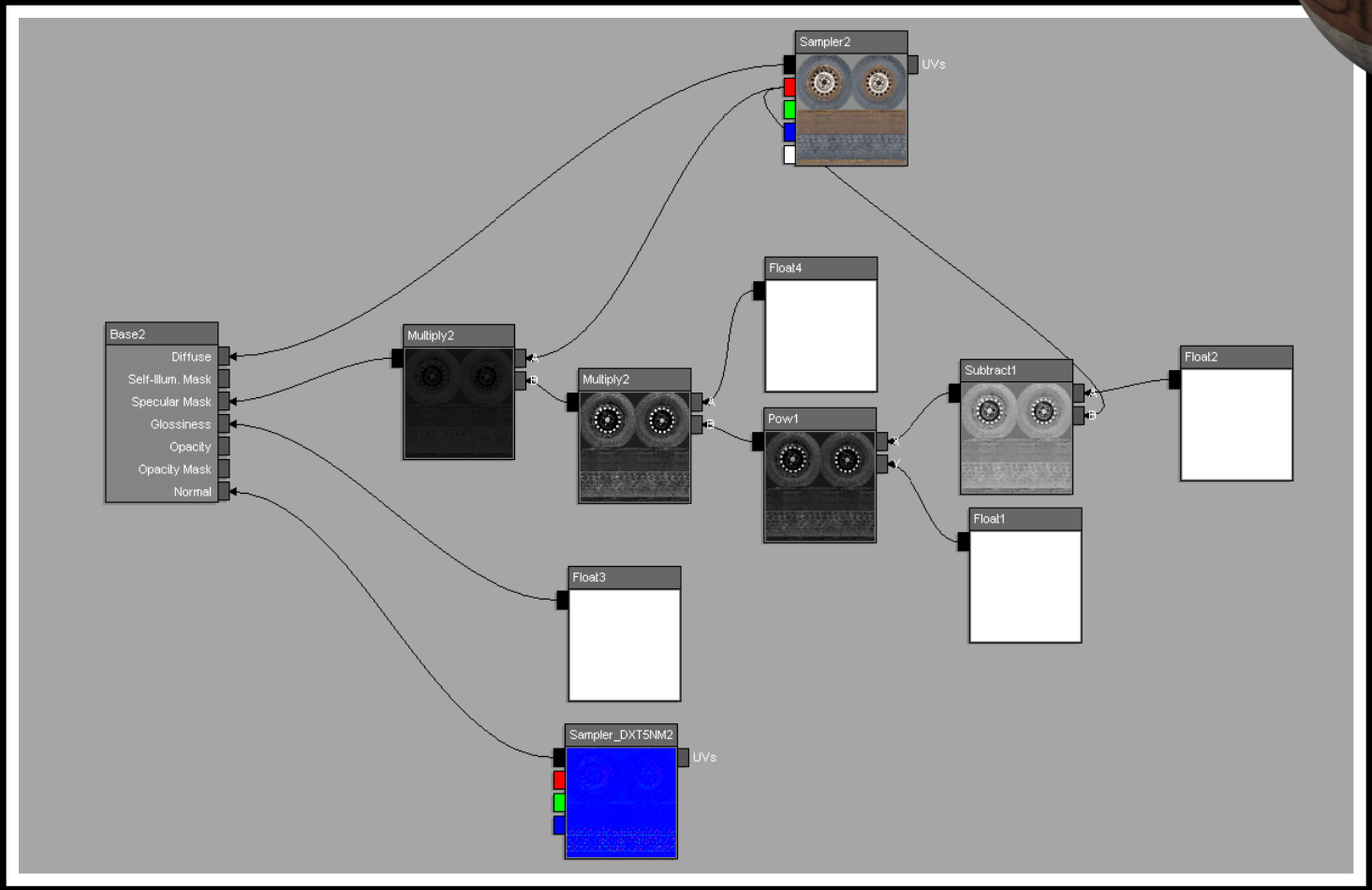
Shader Generation

```
…
float4 CombineStages(VS_OUTPUT Input)
{
 float3 vColor = (float3) 0;
 float fOpacity = 1.0;

 // Diffuse stage
 #ifdef DIFFUSE
  float3 vDiffuse = GetDiffuse(Input);
 #ifdef DIFFUSE_TINT
  vDiffuse *= DIFFUSE_TINT;
 #endif
  vColor = vDiffuse;
 #endif

 // Diffuse lighting stage
 #ifdef LIGHTING_LIGHTMAP
  float3 vLighting = GetStaticLighting(Input);
  vColor *= vLighting;
 #else
 #ifdef LIGHTING_DYNAMIC_LIGHTING
  float3 vLighting = GetDynamicLighting(Input);
  vColor *= vLighting;
 #endif
 #endif
….
```

```
<material>
 <shader>
  <diffuse>
   <map>
    <source>materials/metal/base03.dds</source>
   </map>
  </diffuse>
  <reflection fresnel="1.5">
   <map>
    <source>materials/metal/env.dds</source>
    <type>cube</type>
    <amount>1.0</amount>
    <saturation>0.2</saturation>
   </map>
   <map>
    <source>materials/metal/base03.dds</source>
    <channels>alpha</channels>
    <type>mask</type>
   </map>
  </reflection>
 </shader>
</material>
```

Material

Shader Generation



Demo

Optimizations

Minimize State-Changes

Culling

Draw front-to-back

Instancing

Shader optimizations

questions

clemenskern.de