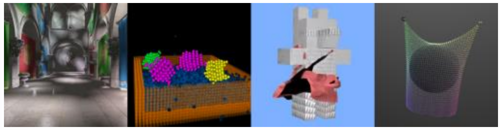


The Graphics Pipeline

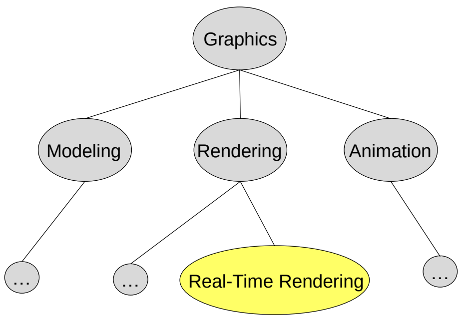
Patrick Cozzi
University of Pennsylvania
CIS 565 - Fall 2016

Agenda

- Graphics Pipeline
- Mapping the Graphics Pipeline to Hardware

2

Graphics Taxonomy



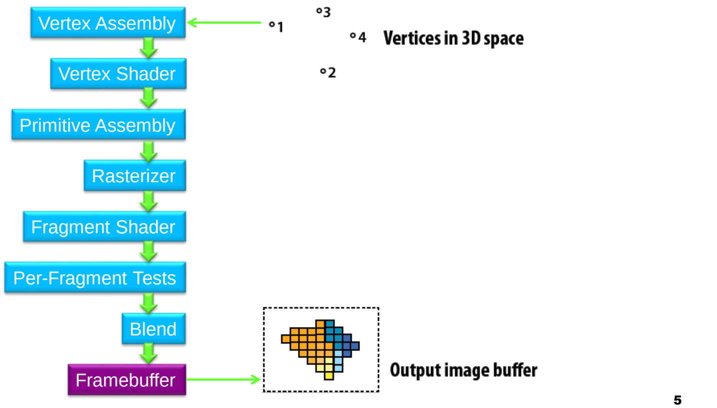
3

Graphics Review: Rendering

- Rendering
  - Goal: Assign color to pixels
- Two Parts
  - Visible surfaces
    - What is in front of what for a given view
  - Shading
    - Simulate the interaction of material and light to produce a pixel color

4

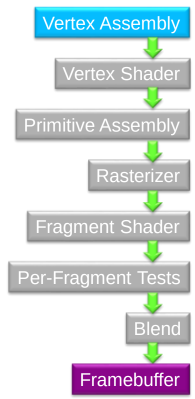
Graphics Pipeline Walkthrough



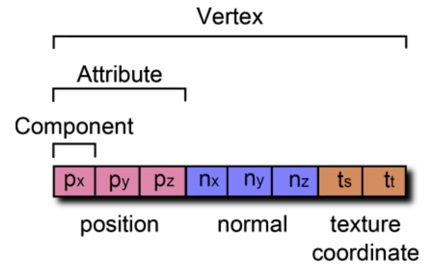
5

Images from http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15869-f11/www/lectures/01_intro.pdf

Vertex Assembly



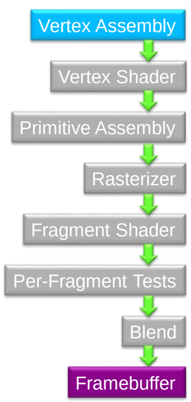
- Pull together a vertex from one or more buffers
- Also called *primitive processing* (GL ES) or *input assembler* (D3D)



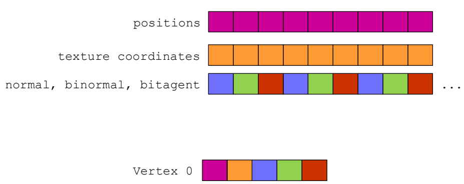
6

Image from http://www.virtualglobebook.com/

Vertex Assembly

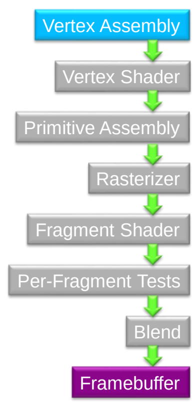


- OpenGL provides lots of flexibility for pulling vertex attributes from different buffers
- For example (no indices):

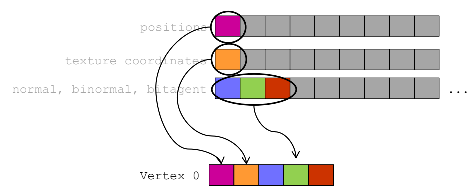


7

Vertex Assembly

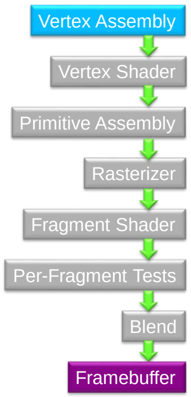


- OpenGL provides lots of flexibility for pulling vertex attributes from different buffers
- For example (no indices):

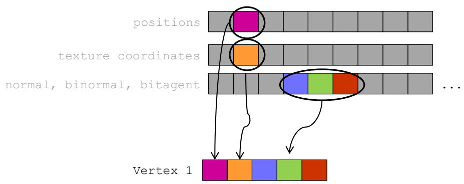


8

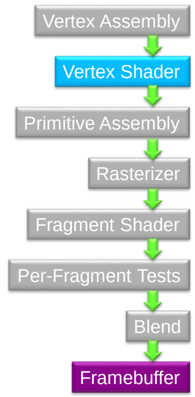
Vertex Assembly



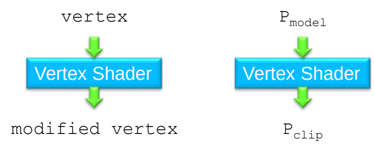
- OpenGL provides lots of flexibility for pulling vertex attributes from different buffers
- For example (no indices):



Vertex Shader

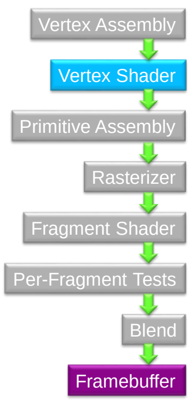


- Transform incoming vertex position from *model* to *clip* coordinates
- Perform additional per-vertex computations; modify, add, or remove attributes passed down the pipeline
- Formerly called the *Transform and Lighting* (T&L) stage. Why?



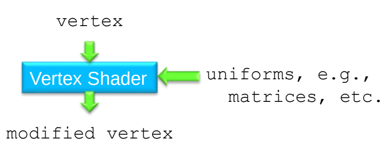
$$P_{clip} = (M_{model\text{-}view\text{-}projection})\,(P_{model})$$

Vertex Shader

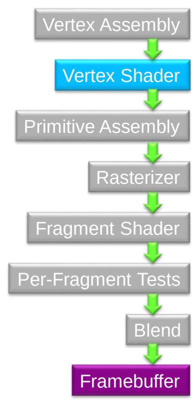


- Model to clip coordinates requires three transforms:
  - model to world
  - world to eye
  - eye to clip
- Use 4x4 matrices passed to the vertex shader as uniforms

$$P_{world} = (M_{model})\,(P_{model})$$
$$P_{eye} = (M_{view})\,(P_{world})$$
$$P_{clip} = (M_{projection})\,(P_{eye})$$



Vertex Shader



Model to world:

$$P_{world} = (M_{model})\,(P_{model})$$

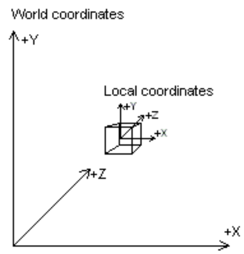


Image from http://msdn.microsoft.com/en-us/library/windows/desktop/bb206365(v=vs.85).aspx

Vertex Shader

Vertex Assembly → **Vertex Shader** → Primitive Assembly → Rasterizer → Fragment Shader → Per-Fragment Tests → Blend → Framebuffer

World to eye:

$$P_{eye} = (M_{view})\ (P_{world})$$

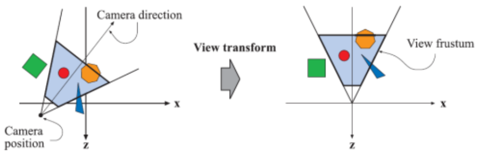


Image from http://www.realtimerendering.com/

Vertex Shader

Vertex Assembly → **Vertex Shader** → Primitive Assembly → Rasterizer → Fragment Shader → Per-Fragment Tests → Blend → Framebuffer

Eye to clip coordinates:

$$P_{clip} = (M_{projection})\ (P_{eye})$$

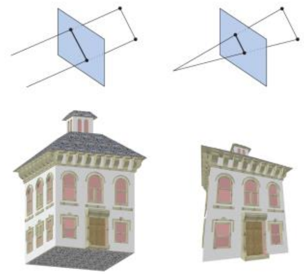


Image from http://www.realtimerendering.com/

Vertex Shader

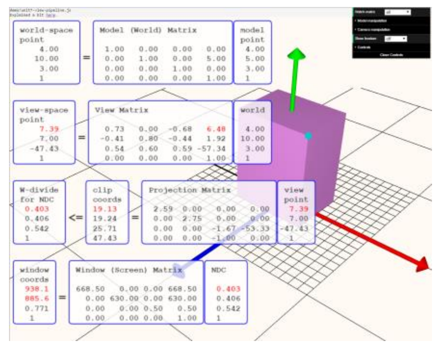
Vertex Assembly → **Vertex Shader** → Primitive Assembly → Rasterizer → Fragment Shader → Per-Fragment Tests → Blend → Framebuffer

- In practice, the model, view, and projection matrices are commonly burnt into one matrix? Why?

$$P_{clip} = (M_{projection})\ (M_{view})\ (M_{model})\ (P_{model})$$
$$P_{clip} = (M_{model\text{-}view\text{-}projection})\ (P_{model})$$

Vertex Shader

Vertex Assembly → **Vertex Shader** → Primitive Assembly → Rasterizer → Fragment Shader → Per-Fragment Tests → Blend → Framebuffer



[Transforms demo]

Screenshot and demo by Eric Haines.

Vertex Shader

Vertex Assembly
Vertex Shader
Primitive Assembly
Rasterizer
Fragment Shader
Per-Fragment Tests
Blend
Framebuffer

- Model to clip coordinate transformation is just one use for the vertex shader.
- Another use: animation.
- How would you implement pulsing?

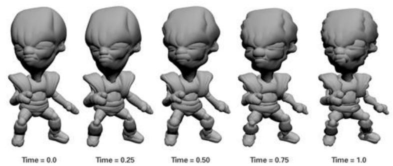
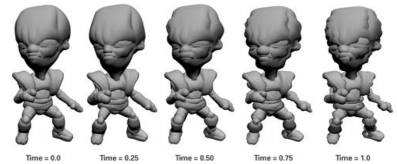


Image from http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter06.html

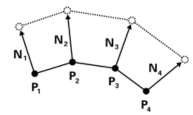
Vertex Shader

Vertex Assembly
Vertex Shader
Primitive Assembly
Rasterizer
Fragment Shader
Per-Fragment Tests
Blend
Framebuffer

- How would you implement pulsing?



- Displace position along surface normal over time



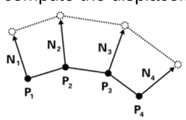
- How do we compute the displacement?

Image from http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter06.html

Vertex Shader

Vertex Assembly
Vertex Shader
Primitive Assembly
Rasterizer
Fragment Shader
Per-Fragment Tests
Blend
Framebuffer

- How do we compute the displacement?



- Consider:

```
float displacement =
  0.5 * (sin(u_time) + 1.0);
```

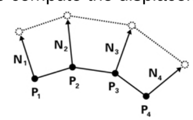
- What are the shortcomings?

Image from http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter06.html

Vertex Shader

Vertex Assembly
Vertex Shader
Primitive Assembly
Rasterizer
Fragment Shader
Per-Fragment Tests
Blend
Framebuffer

- How do we compute the displacement?



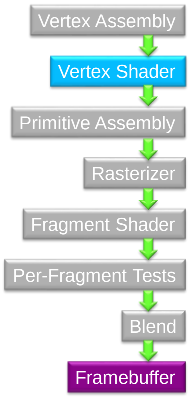
- Consider:

```
float displacement =
  u_scaleFactor * 0.5 *
  (sin(u_frequency * u_time)
  + 1.0);
```

- What are the other shortcomings?

Image from http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter06.html

Vertex Shader



- How do we get the varying bulge?

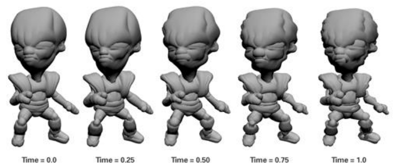
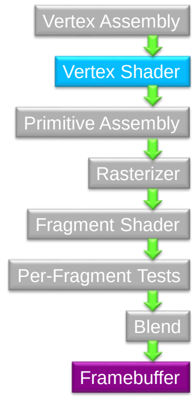
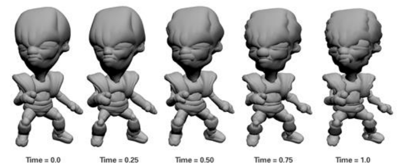


Image from http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter06.html

Vertex Shader



- How do we get the varying bulge?

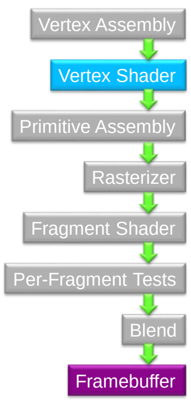


- Consider

```
float displacement =
  u_scaleFactor * 0.5 *
  (sin(position.y * u_frequency *
  u_time) + 1.0);
```

Image from http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter06.html

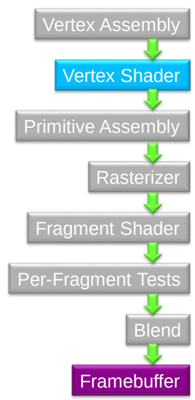
Vertex Shader



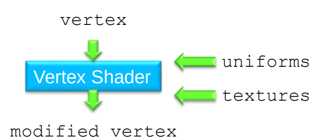
- What varies per-vertex and what does not?

```
float displacement =
  u_scaleFactor * 0.5 *
  (sin(position.y *
  u_frequency * u_time) +
  1.0);
```


Vertex Shader



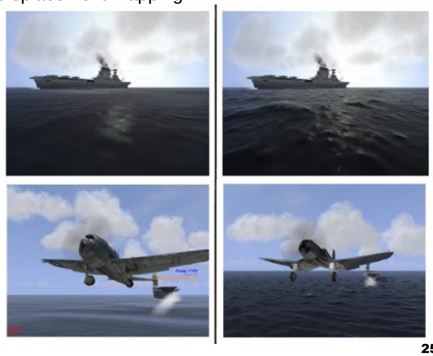
- On all modern GPUs, vertex shaders can read from textures as well as uniform variables.
- What is this useful for?



Vertex Shader

Vertex Assembly
Vertex Shader
Primitive Assembly
Rasterizer
Fragment Shader
Per-Fragment Tests
Blend
Framebuffer

- Example: Textures can provide height maps for displacement mapping

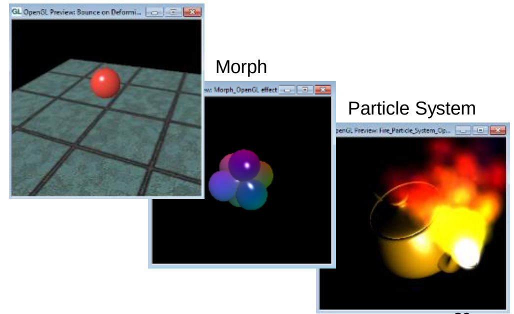


Images from http://developer.nvidia.com/content/vertex-texture-fetch

Vertex Shader

Vertex Assembly
Vertex Shader
Primitive Assembly
Rasterizer
Fragment Shader
Per-Fragment Tests
Blend
Framebuffer

- RenderMonkey Demos

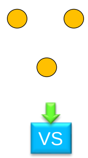


RenderMonkey: http://developer.amd.com/archive/gpu/rendermonkey/pages/default.aspx

Primitive Assembly

Vertex Assembly
Vertex Shader
Primitive Assembly
Rasterizer
Fragment Shader
Per-Fragment Tests
Blend
Framebuffer

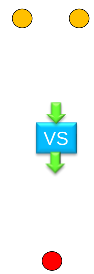
- A vertex shader processes one vertex. *Primitive assembly* groups vertices forming one primitive, e.g., a triangle, line, etc.



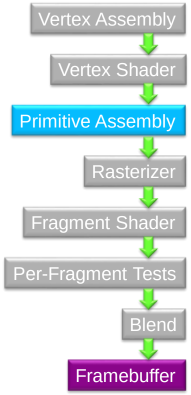
Primitive Assembly

Vertex Assembly
Vertex Shader
Primitive Assembly
Rasterizer
Fragment Shader
Per-Fragment Tests
Blend
Framebuffer

- A vertex shader processes one vertex. *Primitive assembly* groups vertices forming one primitive, e.g., a triangle, line, etc.



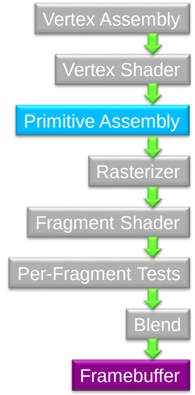
Primitive Assembly



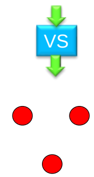
- A vertex shader processes one vertex. *Primitive assembly* groups vertices forming one primitive, e.g., a triangle, line, etc.

29

Primitive Assembly

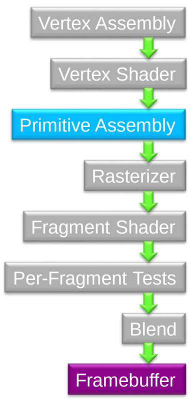


- A vertex shader processes one vertex. *Primitive assembly* groups vertices forming one primitive, e.g., a triangle, line, etc.

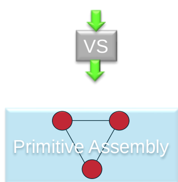


30

Primitive Assembly

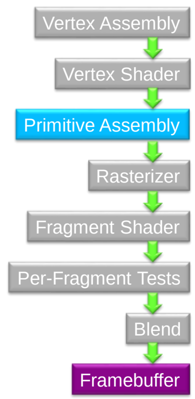


- A vertex shader processes one vertex. *Primitive assembly* groups vertices forming one primitive, e.g., a triangle, line, etc.

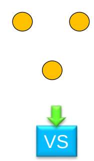


31

Primitive Assembly

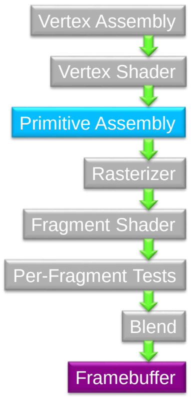


- A vertex shader processes one vertex. *Primitive assembly* groups vertices forming one primitive, e.g., a triangle, line, etc.

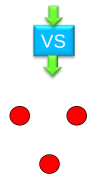


32

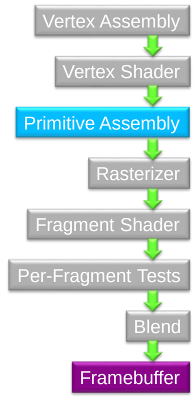
Primitive Assembly



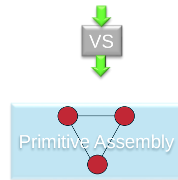
- A vertex shader processes one vertex. *Primitive assembly* groups vertices forming one primitive, e.g., a triangle, line, etc.



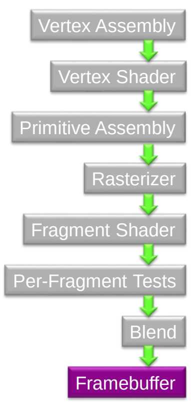
Primitive Assembly



- A vertex shader processes one vertex. *Primitive assembly* groups vertices forming one primitive, e.g., a triangle, line, etc.



Perspective Division and Viewport Transform



- There are a series of stages between primitive assembly and rasterization.
  - *Perspective division*

    $P_{ndc} = (P_{clip}).xyz \ / \ (P_{clip}).w$
  - *Viewport transform*

    $P_{window} = (M_{viewport-transform})(P_{ndc})$

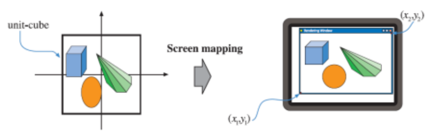
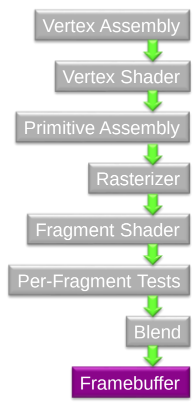


Image from http://www.realtimerendering.com/

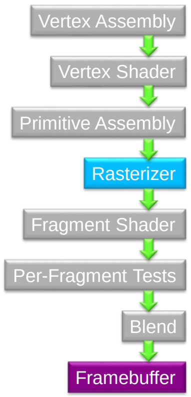
Clipping



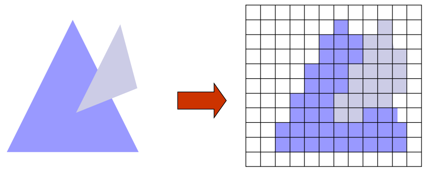
- There are a series of stages between primitive assembly and rasterization.
  - *Clipping*

Image from http://www.realtimerendering.com/

Rasterization

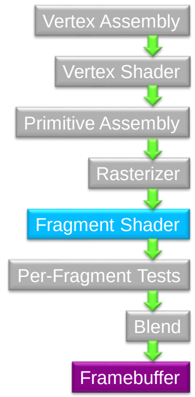


- Determine what pixels a primitive overlaps

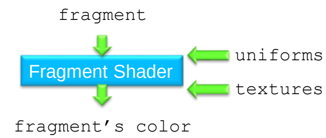


- How would you implement this?
- What about aliasing?
- What happens to non-position vertex attributes during rasterization?
- What is the triangle-to-fragment ratio?

Fragment Shader

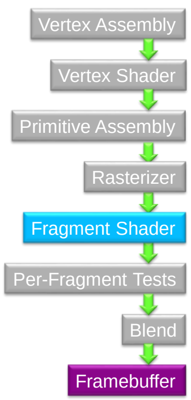


- Shades the fragment by simulating the interaction of light and material
- Loosely, the combination of a fragment shader and its uniform inputs is a *material*
- Also called a *Pixel Shader* (D3D)

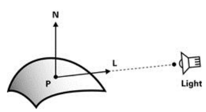


- Fragment shaders can be computationally intense
- What exactly is the fragment input?
- What are examples of useful uniforms? Useful textures?

Fragment Shader



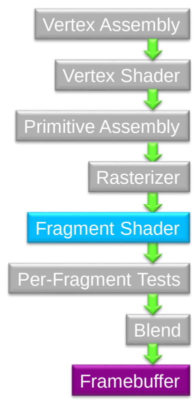
- Example: Blinn-Phong Lighting



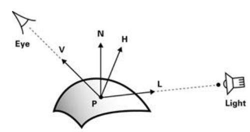

```
float diffuse =
  max(dot(N, L), 0.0);
```


Image from http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter05.html

Fragment Shader



- Example: Blinn-Phong Lighting




```
float specular =
  max(pow(dot(H, N),
  u_shininess), 0.0);
```

- Why not evaluate per-vertex and interpolate during rasterization?

Image from http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter05.html

Fragment Shader

Vertex Assembly → Vertex Shader → Primitive Assembly → Rasterizer → Fragment Shader → Per-Fragment Tests → Blend → Framebuffer

- Per-fragment vs. per-vertex lighting
- Which is which?

41

Screenshot and demo by Eric Haines and Gundega Dekena. Udacity Interactive 3D Graphics.

Fragment Shader

Vertex Assembly → Vertex Shader → Primitive Assembly → Rasterizer → Fragment Shader → Per-Fragment Tests → Blend → Framebuffer

- Example: Texture Mapping

42

Image from http://www.realtimerendering.com/

Fragment Shader

Vertex Assembly → Vertex Shader → Primitive Assembly → Rasterizer → Fragment Shader → Per-Fragment Tests → Blend → Framebuffer

- Lighting and texture mapping

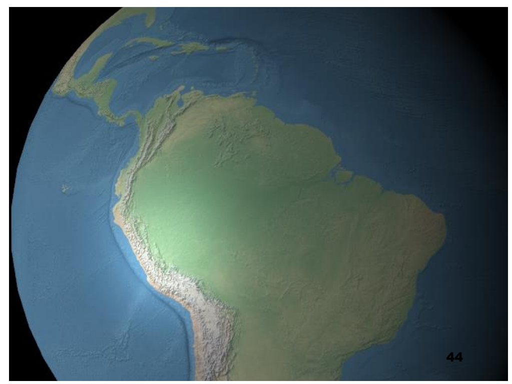
43

Image from http://www.virtualglobebook.com/

Fragment Shader

Vertex Assembly → Vertex Shader → Primitive Assembly → Rasterizer → Fragment Shader → Per-Fragment Tests → Blend → Framebuffer

- Example: Bump mapping

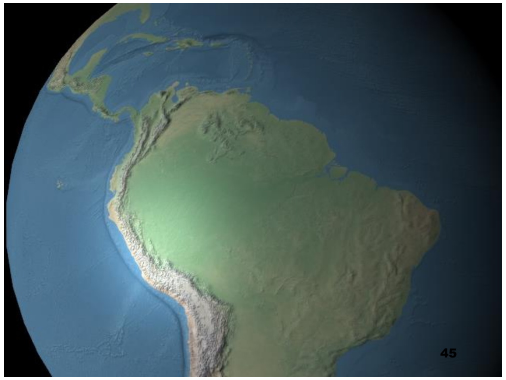


44

Fragment Shader

Vertex Assembly
↓
Vertex Shader
↓
Primitive Assembly
↓
Rasterizer
↓
Fragment Shader
↓
Per-Fragment Tests
↓
Blend
↓
Framebuffer

- Example: Bump mapping



Fragment Shader

Vertex Assembly
↓
Vertex Shader
↓
Primitive Assembly
↓
Rasterizer
↓
Fragment Shader
↓
Per-Fragment Tests
↓
Blend
↓
Framebuffer

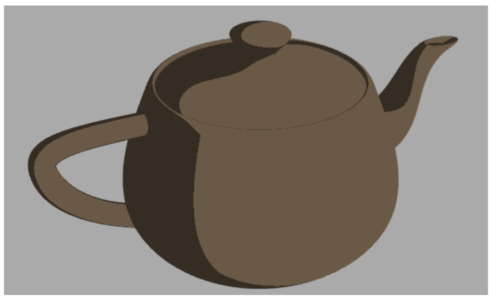
- Example: Specular map

Screenshot and demo by Eric Haines and Gundega Dekena. Udacity Interactive 3D Graphics.

Fragment Shader

Vertex Assembly
↓
Vertex Shader
↓
Primitive Assembly
↓
Rasterizer
↓
Fragment Shader
↓
Per-Fragment Tests
↓
Blend
↓
Framebuffer

- Example: Non-Photorealistic Rendering (NPR)



Screenshot and demo by Eric Haines and Gundega Dekena. Udacity Interactive 3D Graphics.

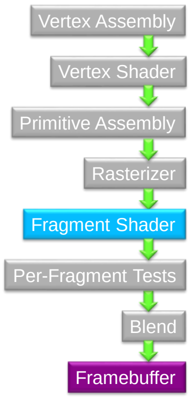
Fragment Shader

Vertex Assembly
↓
Vertex Shader
↓
Primitive Assembly
↓
Rasterizer
↓
Fragment Shader
↓
Per-Fragment Tests
↓
Blend
↓
Framebuffer

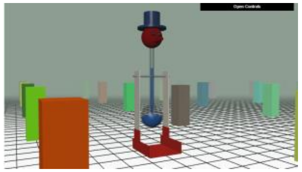
- Example: Reflection mapping

Screenshot and demo by Eric Haines and Gundega Dekena. Udacity Interactive 3D Graphics.

Fragment Shader



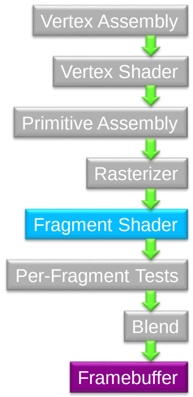
- More examples



Fog

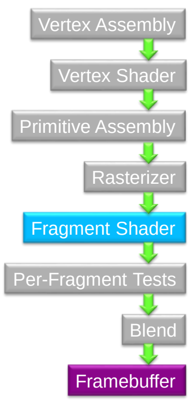
Screenshot and demo by Eric Haines and Gundega Dekena. Udacity Interactive 3D Graphics.

Fragment Shader



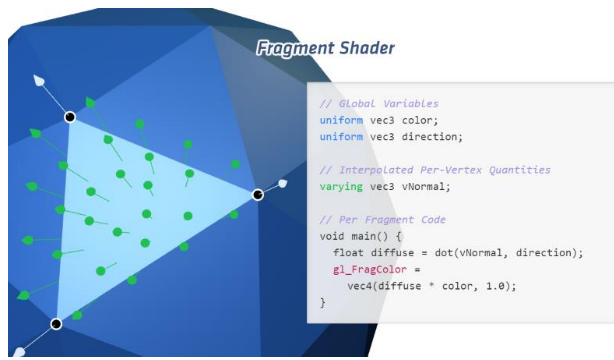
- A fragment shader can output color, but what else would be useful?

Fragment Shader



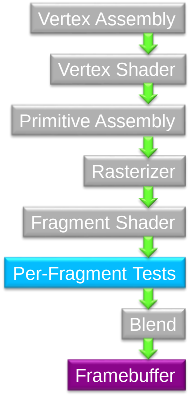
- A fragment shader can output color, but what else would be useful?
  - Discard the fragment. Why?
  - Depth. Why?
  - Multiple colors. Why?

Vertex and Fragment Shader Examples



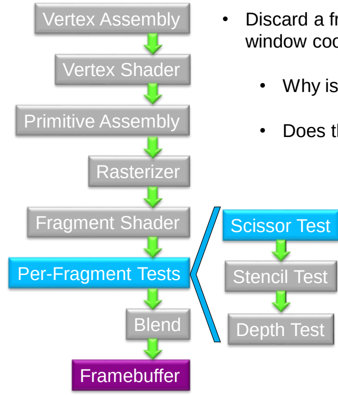
By Steven Wittens, @unconed

Per-Fragment Tests



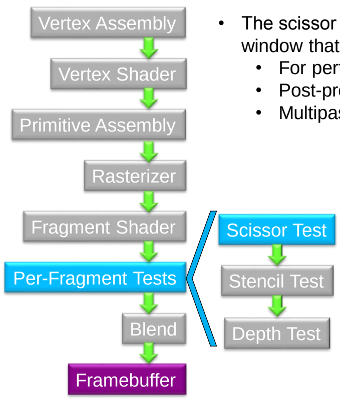
- A fragment must go through a series of tests to make to the framebuffer
  - What tests are useful?

Scissor Test



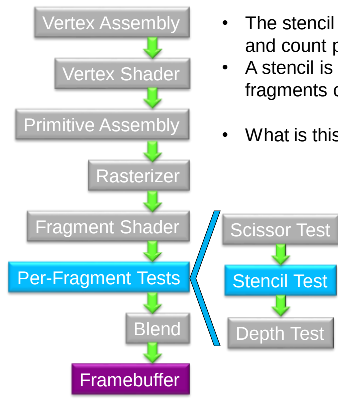
- Discard a fragment if it is within a rectangle defined in window coordinates
  - Why is this useful?
  - Does this need to happen after fragment shading?

Scissor Test



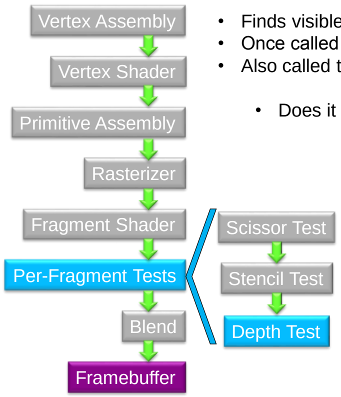
- The scissor test is useful for “scissoring out” parts of the window that do not need to be shaded
  - For performance. Why?
  - Post-processing effects
  - Multipass rendering

Stencil Test

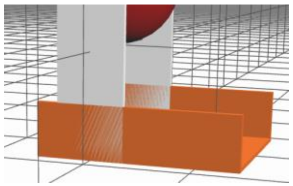


- The stencil test can discard arbitrary areas of the window, and count per-fragment
- A stencil is written to the stencil buffer, and later fragments can be tested against this buffer
- What is this useful for?

Depth Test



- Finds visible surfaces
- Once called "ridiculously expensive"
- Also called the *z-test*
  - Does it need to be after fragment shading?



z-fighting

Screenshot and demo by Eric Haines and Gundega Dekena. Udacity Interactive 3D Graphics.

Depth Test

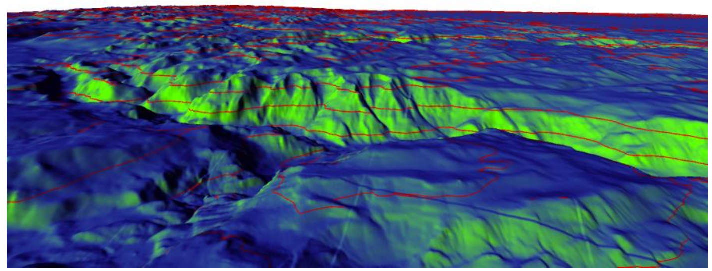
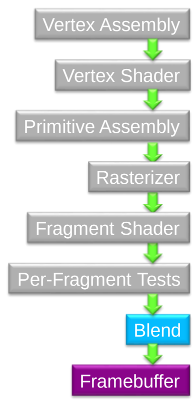


Image from http://www.virtualglobebook.com/

Depth Test

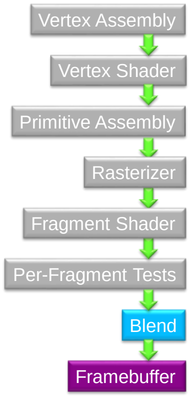
Image from http://www.virtualglobebook.com/

Blending



- Combine fragment color with framebuffer color
  - Can weight each color
  - Can use different operations: +, -, etc.
- Why is this useful?

Blending



- Example: Translucency
- Additive Blending

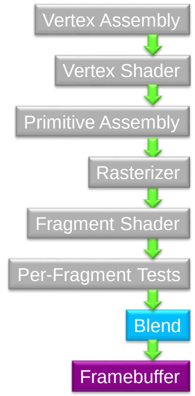
```
Cdest = (Csource.rgb)(Csource.a) + (Cdest.rgb);
```

- Alpha Blending

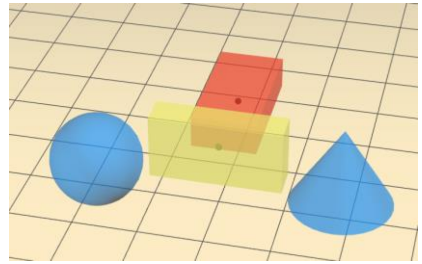
```
Cdest = (Csource.rgb)(Csource.a) +
   (Cdest.rgb) (1 - Csource.a);
```

Image from http://http.developer.nvidia.com/GPUGems/gpugems_ch06.html

Blending



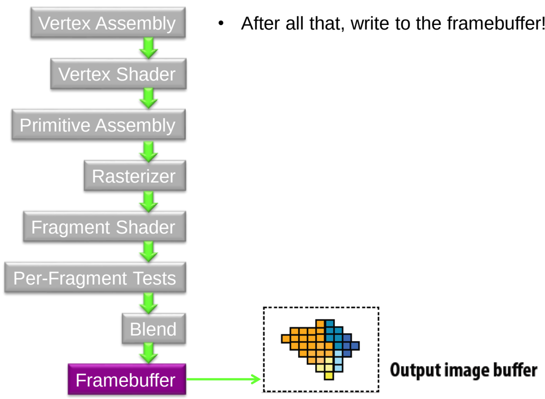
- Alpha Blending sorting



Screenshot and demo by Eric Haines and Gundega Dekena. Udacity Interactive 3D Graphics.

Graphics Pipeline Walkthrough

- After all that, write to the framebuffer!



Images from http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15869-f11/www/lectures/01_intro.pdf

Example Performance Analysis

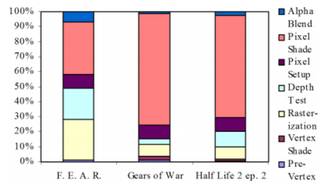


Figure 13: *End-to-End Average Time Breakdowns: shows the average time spent in each rendering stage for the three games*

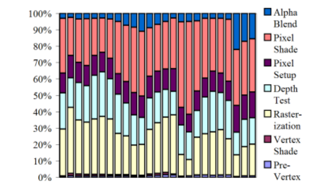


Figure 14: *F.E.A.R. per-frame time breakdowns: this chart shows the time spent in each rendering stage for 25 widely spaced frames of F.E.A.R., which show considerable load variation.*

Images from http://www.student.chemia.uj.edu.pl/~mrozek/USl/wyklad/Nowe_konstrukcje/Siggraph_Larrabee_paper.pdf

Evolution of the Programmable Graphics Pipeline

- Pre GPU
- Fixed function GPU
- Programmable GPU
- Unified Shader Processors

Early 90s – Pre GPU

Wolfenstein 3D, 1992

Doom I, 1993

- Interactive software rendering (no GPUs yet)
- NOTE: SGI was building interactive rendering supercomputers, but this was beginning of interactive 3D graphics on PC

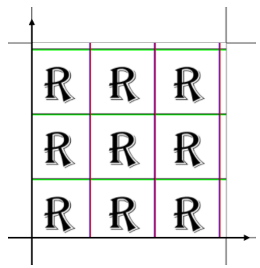
Slide from http://s09.idav.ucdavis.edu/talks/01-BPS-SIGGRAPH09-mhouston.pdf

Why GPUs?

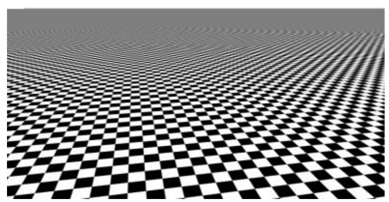
- Exploit Parallelism
  - Pipeline parallel
  - Data-parallel
  - CPU and GPU executing in parallel
- Hardware:
  - Texture filtering, rasterization, alpha blending, …
  - MAD, sqrt, ...

Texture Hardware

- Software is 12-40x slower (Larrabee)



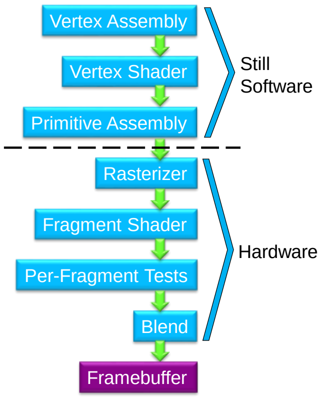
Wrap modes



Texture filtering

Screenshot and demo by Eric Haines and Gundega Dekena. Udacity Interactive 3D Graphics.

3dfx Voodoo (1996)



In hardware:

- Fixed-function rasterization, texture mapping, depth testing, etc.
- 4 - 6 MB memory
- PCI bus
- $299

Image from http://www.thedodgegarage.com/3dfx/v1.htm

Aside: Mario Kart 64

- High fragment load / low vertex load

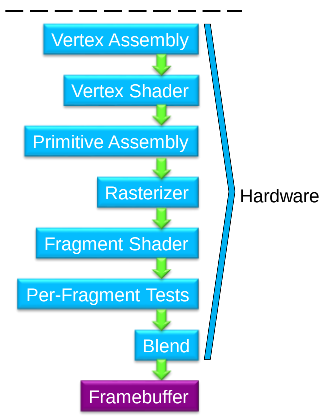
Image from http://www.gamespot.com/users/my_shoe/

Aside: Mario Kart Wii

- High fragment load / low vertex load?

Image from http://wii.ign.com/dor/objects/949580/mario-kart-wii/images/

NVIDIA GeForce 256 (1999)



In hardware:

- Fixed-function vertex shading (T&L)
- Multi-texturing: bump maps, light maps, etc.
- 10 million polygons per second
- Direct3D 7
- AGP bus

Image from http://en.wikipedia.org/wiki/File:VisionTek_GeForce_256.jpg

NVIDIA GeForce 3 (2001)

Vertex Assembly → Vertex Shader → Primitive Assembly → Rasterizer → Fragment Shader → Per-Fragment Tests → Blend → Framebuffer

Hardware

- Optionally bypass fixed-function T&L with a programmable vertex shader
- Optionally bypass fixed-function fragment shading with a programmable fragment shader (register combiners)
- Many programming limits
- Direct3D 8

- Pentium IV – 20 stages
- GeForce 3 – 600-800 stages

Fixed-function stage

Programmable stage

73

NVIDIA GeForce 6 (2004)

Vertex Assembly → Vertex Shader → Primitive Assembly → Rasterizer → Fragment Shader → Per-Fragment Tests → Blend → Framebuffer

textures

Hardware

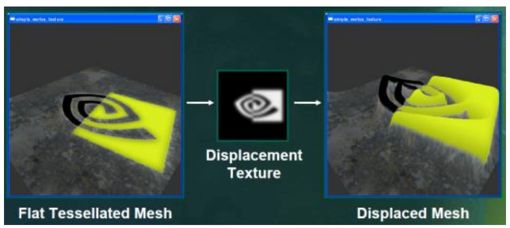
- Much better programmable fragment shaders
- Vertex shader can read textures
- Dynamic branches
- Multiple render targets
- PCIe bus
- OpenGL 2 / Direct3D 9

74

NVIDIA GeForce 6 (2004)

Vertex Assembly → Vertex Shader → Primitive Assembly → Rasterizer → Fragment Shader → Per-Fragment Tests → Blend → Framebuffer

- Vertex shader can read textures
- Dynamic branches
- Multiple render targets
- PCIe bus
- OpenGL 2 / Direct3D 9



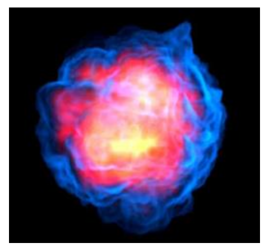
75

Image from ftp://download.nvidia.com/developer/presentations/2004/GPU_Jackpot/Shader_Model_3.pdf

NVIDIA GeForce 6 (2004)

Vertex Assembly → Vertex Shader → Primitive Assembly → Rasterizer → Fragment Shader → Per-Fragment Tests → Blend → Framebuffer

- Vertex shader can read textures
- Dynamic branches
- Multiple render targets
- PCIe bus
- OpenGL 2 / Direct3D 9



76

Image from ftp://download.nvidia.com/developer/presentations/2004/GPU_Jackpot/Shader_Model_3.pdf

Dynamic Branches

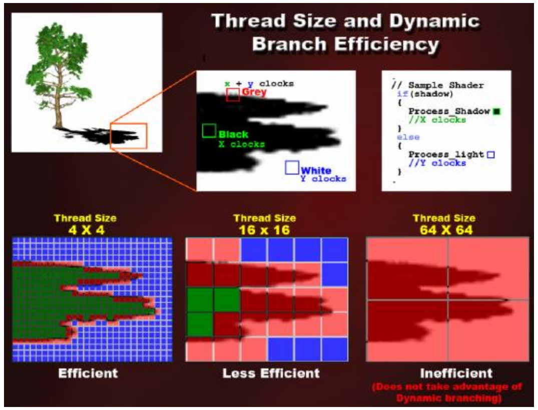


Image from http://developer.amd.com/media/gpu_assets/03_Clever_Shader_Tricks.pdf

Dynamic Branches

- For best performance, fragment shader dynamic branches should be coherent in screen-space
- How does this relate to warp partitioning in CUDA?

NVIDIA GeForce 6 (2004)

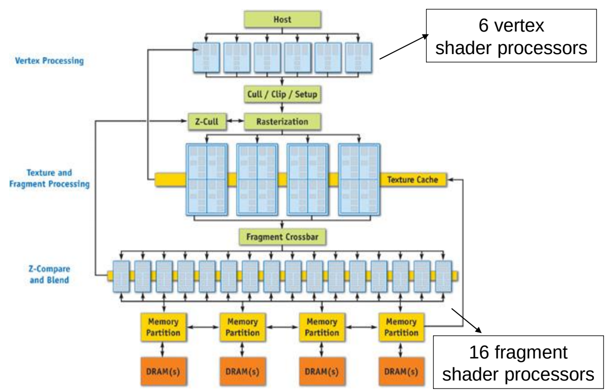
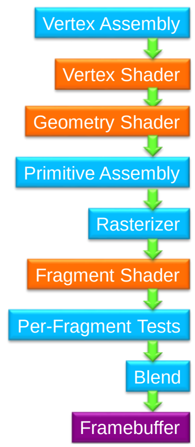


Image from http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter30.html

NVIDIA GeForce 8 (2006)



- Ground-up GPU redesign
- Geometry Shaders
- Transform-feedback
- OpenGL 3 / Direct3D 10
- Unified shader processors
- Support for GPU Compute

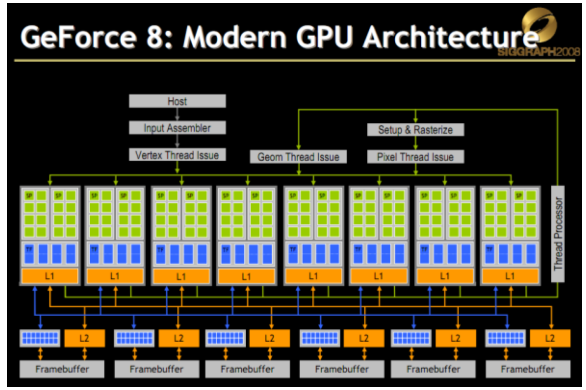
Geometry Shaders

Figure 5: From left — render to cube map, particle system, instancing, shadow volume, displacement mapping.

81

Image from David Blythe : http://download.microsoft.com/download/f/2/d/f2d5ee2c-b7ba-4cd0-9686-b6508b5479a1/direct3d10_web.pdf

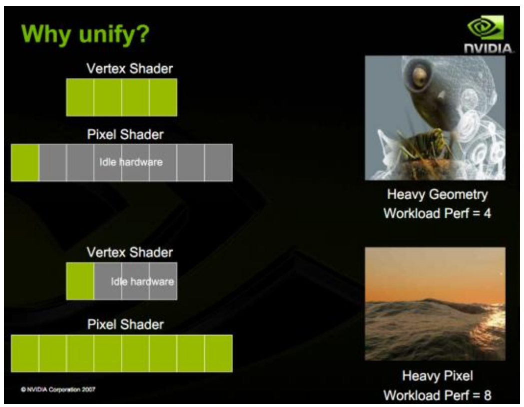
NVIDIA G80 Architecture



82

Slide from http://s08.idav.ucdavis.edu/luebke-nvidia-gpu-architecture.pdf

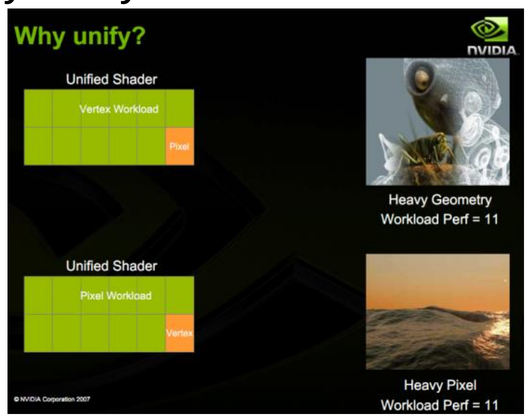
Why Unify Shader Processors?



83

Slide from http://s08.idav.ucdavis.edu/luebke-nvidia-gpu-architecture.pdf

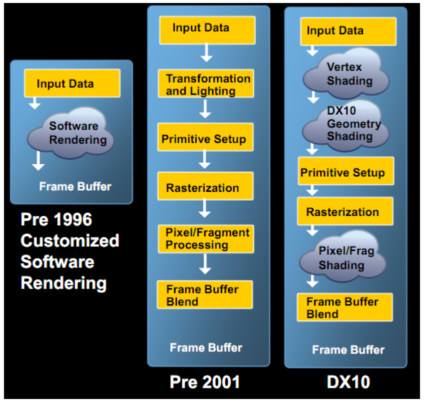
Why Unify Shader Processors?



84

Slide from http://s08.idav.ucdavis.edu/luebke-nvidia-gpu-architecture.pdf

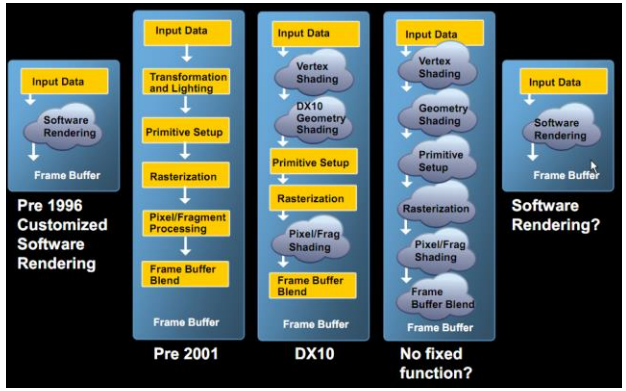
Evolution of the Programmable Graphics Pipeline



85

Slide from Mike Houston: http://s09.idav.ucdavis.edu/talks/01-BPS-SIGGRAPH09-mhouston.pdf

Evolution of the Programmable Graphics Pipeline



86

Slide from Mike Houston: http://s09.idav.ucdavis.edu/talks/01-BPS-SIGGRAPH09-mhouston.pdf