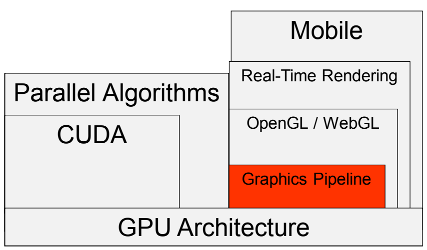
The Graphics Pipeline

Patrick Cozzi
University of Pennsylvania
CIS 565 - Fall 2012

Announcements

- 10/29 – Eric Lengyel Game Engine Architecture Guest Lecture
- 10/30 – Class in SIG lab
  - Instead of class on 10/31
- Build vs. buy

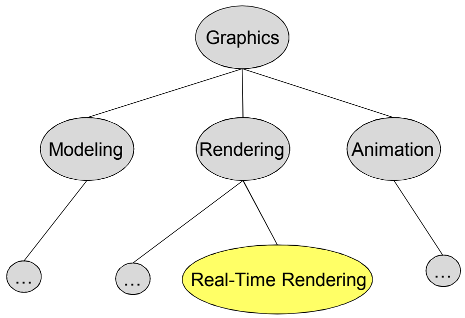
Course Contents



Agenda

- Brief Graphics Review
- Graphics Pipeline
- Mapping the Graphics Pipeline to Hardware

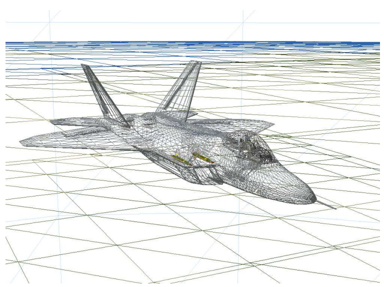
Graphics Review



Graphics Review: Modeling

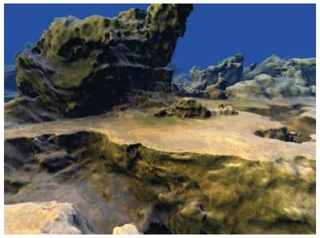
- Modeling
  - Polygons vs. triangles
    - How do we store a triangle mesh?
  - Implicit Surfaces
  - …

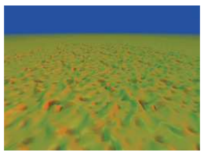
Triangles

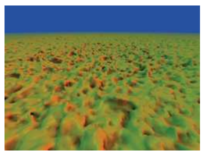


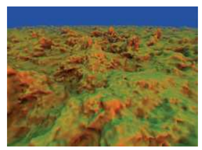
Triangles

Implicit Surfaces





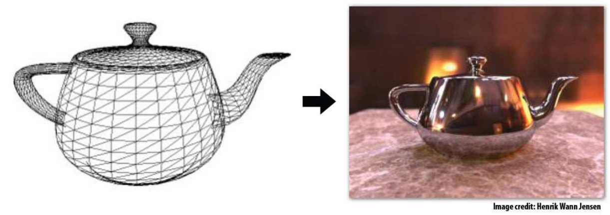




9

Images from http://http.developer.nvidia.com/GPUGems3/gpugems3_ch01.htm

Graphics Review: Rendering



Model of a scene:
3D surface geometry (e.g., triangle mesh)
surface materials
lights
camera

Image

How does each triangle contribute to each pixel in the image?

Kayvon Fatahalian   CMU 15-869, Fall 2011

10

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

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

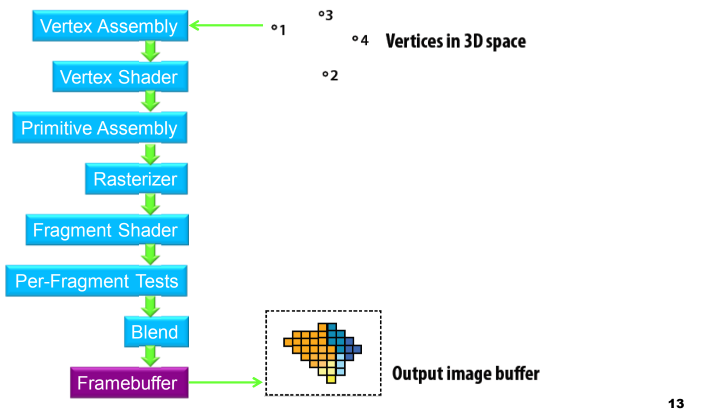
11

Graphics Review: Animation

- Move the camera and/or agents, and re-render the scene
  - In less than 16.6 ms (60 fps)

12

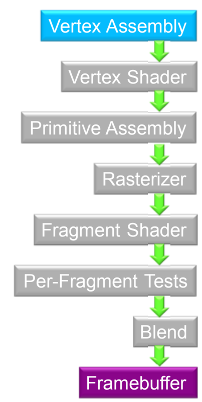
Graphics Pipeline Walkthrough



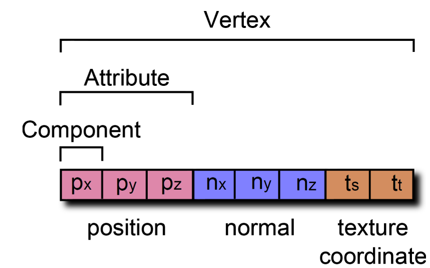
13

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

Vertex Assembly



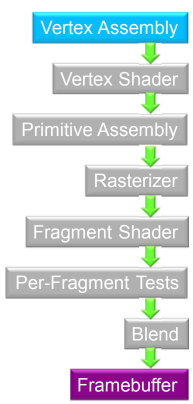
- Pull together a vertex from one or more buffers
- Also called *primitive processing* (GL ES) or *input assembler* (D3D)



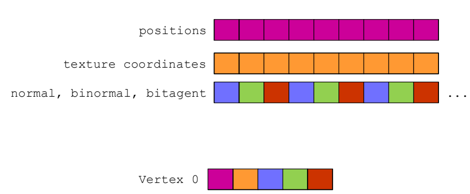
14

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

Vertex Assembly

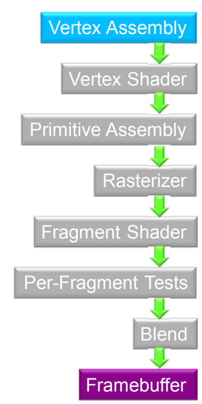


- OpenGL provides lots of flexibility for pulling vertex attributes from different buffers
- For example (no indices):

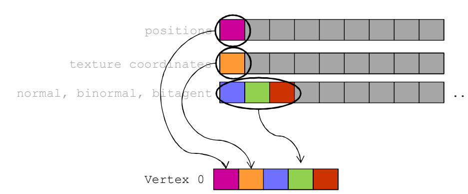


15

Vertex Assembly



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

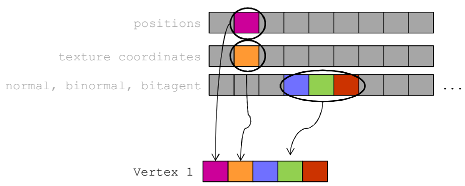


16

Vertex Assembly

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

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



Vertex Shader

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

- 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?

vertex → Vertex Shader → modified vertex

P_{model} → Vertex Shader → P_{clip}

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

Vertex Shader

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

- 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 → Vertex Shader ← uniforms, e.g., matrices, etc.

Vertex Shader → modified vertex

Vertex Shader

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

Model to world:

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

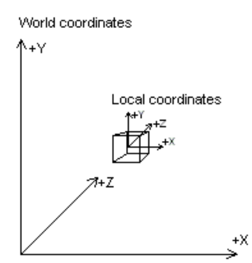


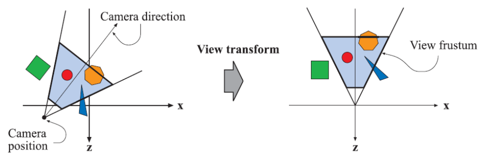
Image from http://msdn.microsoft.com/en-us/library/windows/desktop/bb206365(v=vs.85).aspx

Vertex Shader

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

World to eye:

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



21

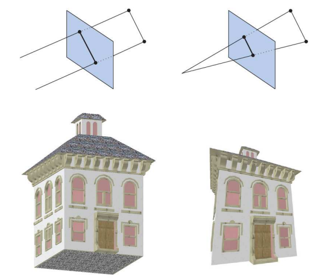
Image from http://www.realtimerendering.com/

Vertex Shader

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

Eye to clip coordinates:

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



22

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?

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

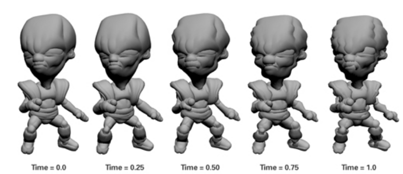
23

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?



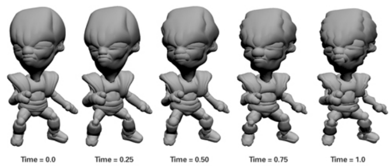
24

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

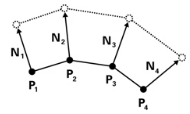
Vertex Shader

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



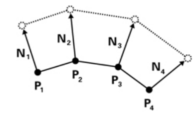
- How do we compute the displacement?

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);
```

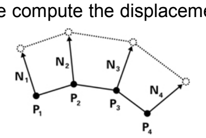
- What are the shortcomings?

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

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

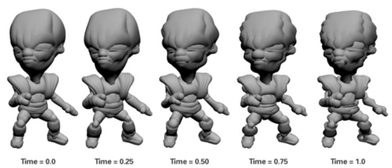
- How do we get the varying bulge?

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 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

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

- 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

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

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

vertex → Vertex Shader → modified vertex

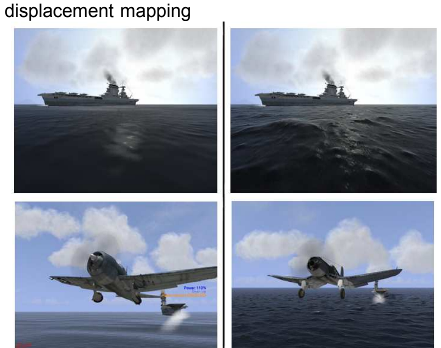
uniforms → Vertex Shader

textures → Vertex Shader

Vertex Shader

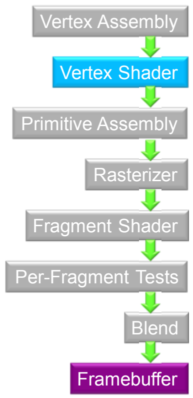
- Vertex Assembly
- Vertex Shader
- Primitive Assembly
- Rasterizer
- Fragment Shader
- Per-Fragment Tests
- Blend
- Framebuffer

- Example: Textures can provide height maps for displacement mapping



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

Vertex Shader



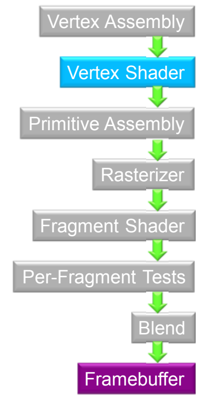
- Technology preview: vertex shaders are becoming available to CSS on the web as *CSS shaders*

Demo

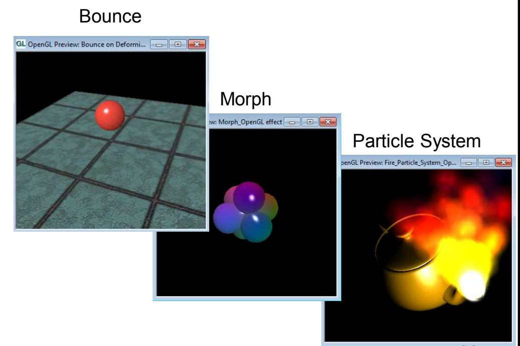
http://www.adobe.com/devnet/html5/articles/css-shaders.html

More info on CSS shaders: https://dvcs.w3.org/hg/FXTF/raw-file/tip/custom/index.html

Vertex Shader

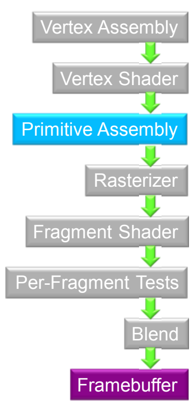


- RenderMonkey Demos

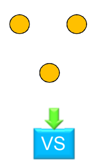


RenderMonkey: http://developer.amd.com/archive/gpu/rendermonkey/pages/default.aspx

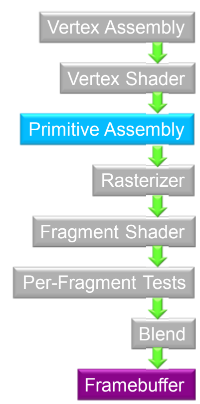
Primitive Assembly



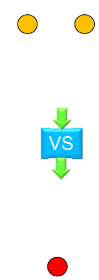
- A vertex shader processes one vertex. *Primitive assembly* groups vertices forming one primitive, e.g., a triangle, line, etc.



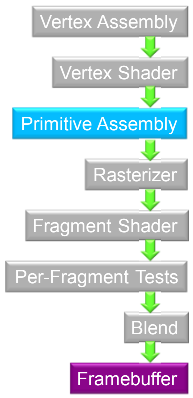
Primitive Assembly



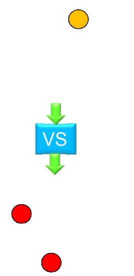
- A vertex shader processes one vertex. *Primitive assembly* groups vertices forming one primitive, e.g., a triangle, line, etc.



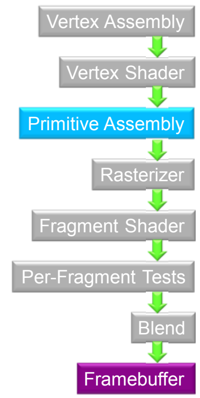
Primitive Assembly



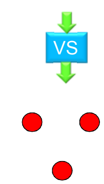
- A vertex shader processes one vertex. *Primitive assembly* groups vertices forming one primitive, e.g., a triangle, line, etc.



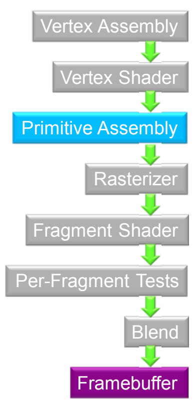
Primitive Assembly



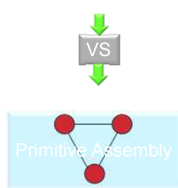
- A vertex shader processes one vertex. *Primitive assembly* groups vertices forming one primitive, e.g., a triangle, line, etc.



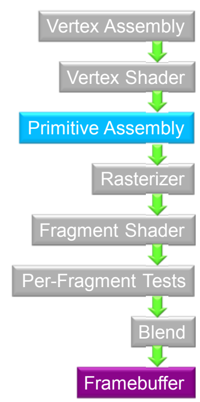
Primitive Assembly



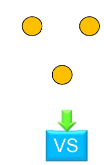
- A vertex shader processes one vertex. *Primitive assembly* groups vertices forming one primitive, e.g., a triangle, line, etc.



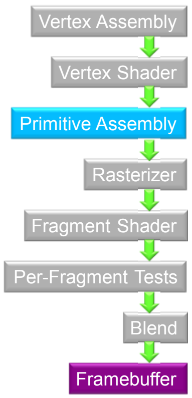
Primitive Assembly



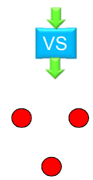
- A vertex shader processes one vertex. *Primitive assembly* groups vertices forming one primitive, e.g., a triangle, line, etc.



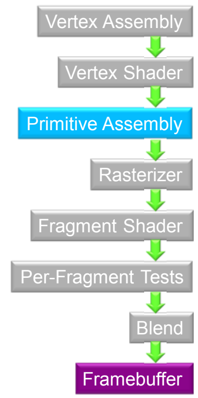
Primitive Assembly



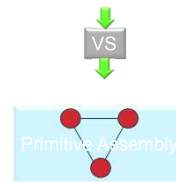
- A vertex shader processes one vertex. *Primitive assembly* groups vertices forming one primitive, e.g., a triangle, line, etc.



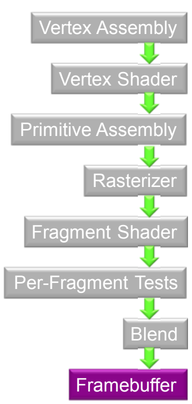
Primitive Assembly



- 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*

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

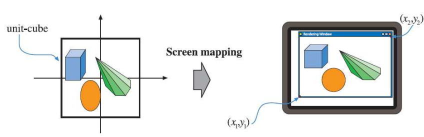
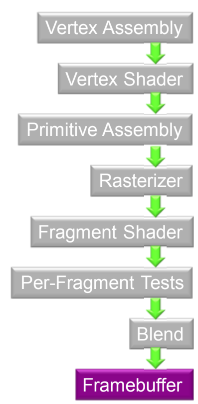


Image from http://www.realtimerendering.com/

Clipping



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

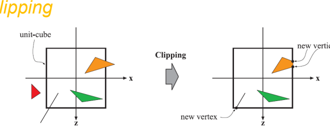
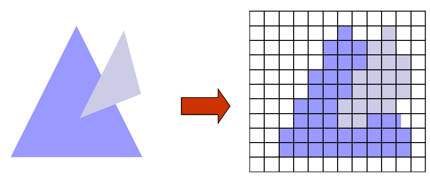


Image from http://www.realtimerendering.com/

Rasterization

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

- 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

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

- 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)

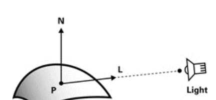
```
fragment
Fragment Shader   <- uniforms
                  <- textures
fragment's color
```

- What exactly is the fragment input?
- What are examples of useful uniforms? Useful textures?

Fragment Shader

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

- Example: Blinn-Phong Lighting



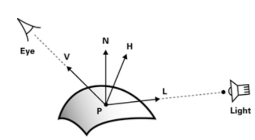

```
float diffuse =
  max(dot(N, L), 0.0);
```


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

Fragment Shader

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

- 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?

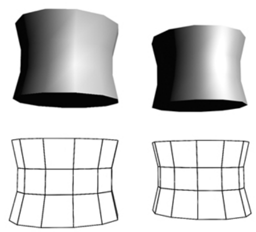


Image from http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter05.html

Fragment Shader

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

* Effects of tessellation on per-vertex lighting

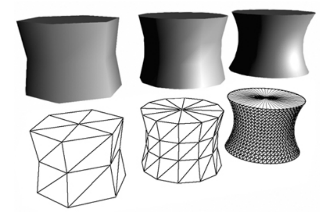


Image from http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter05.html

Fragment Shader

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

* Another example: Texture Mapping

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

Fragment Shader

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

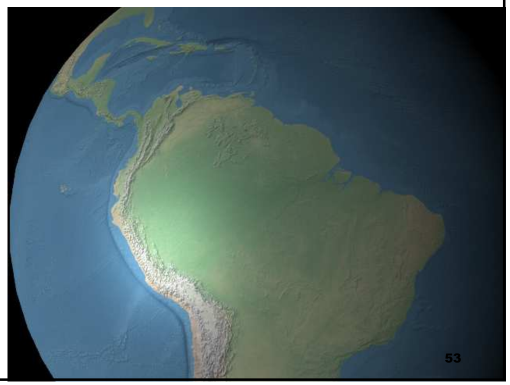
* Lighting and texture mapping

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

Fragment Shader

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

- Another example: Bump mapping

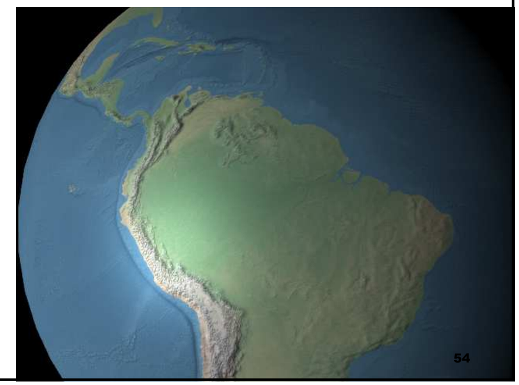


53

Fragment Shader

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

- Another example: Bump mapping



54

Fragment Shader

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

- Fragment shaders can be computationally intense

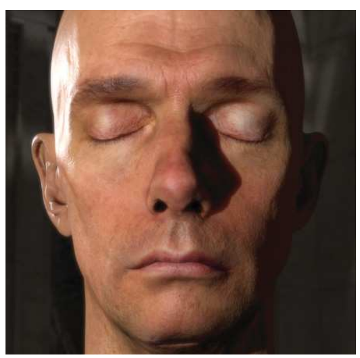


Image from http://http.developer.nvidia.com/GPUGems3/gpugems3_ch14.html

55

Fragment Shader

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

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

56

Fragment Shader

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

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

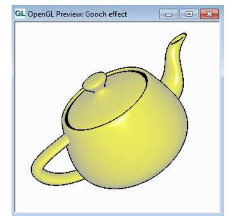
57

Fragment Shader

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

- RenderMonkey Demos

NPR



Glass

58

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

Per-Fragment Tests

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

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

59

Scissor Test

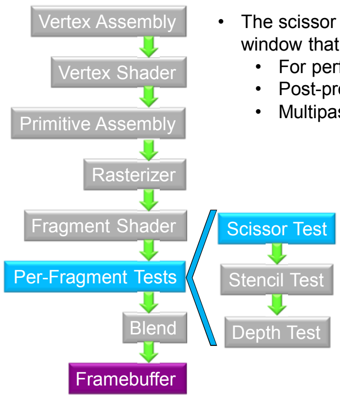
Vertex Assembly
Vertex Shader
Primitive Assembly
Rasterizer
Fragment Shader
Per-Fragment Tests
Blend
Framebuffer

Scissor Test
Stencil Test
Depth 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?

60

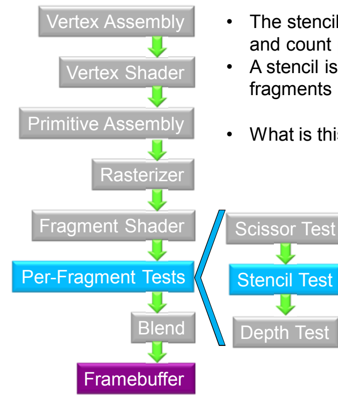
Scissor Test



- 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

61

Stencil Test

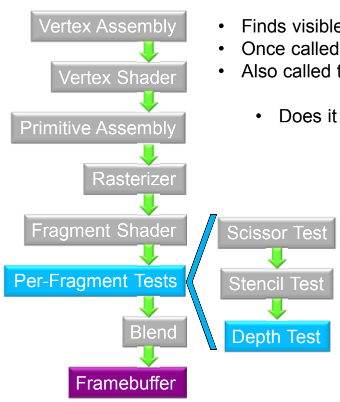


- The stencil test can discard arbitrary areas of the window, and count per-fragment
- A stencil is written to the stencil buffer, and later fragments can be tested against this buffer

- What is this useful for?

62

Depth Test

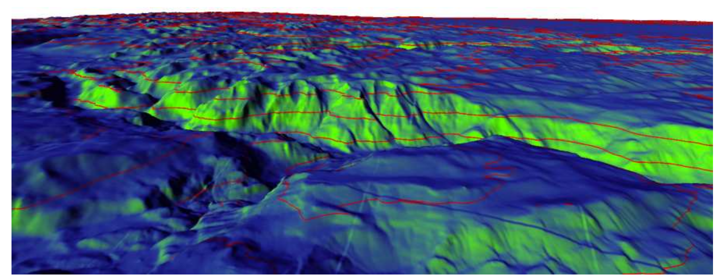


- Finds visible surfaces
- Once called “ridiculously expensive”
- Also called the *z-test*

  - Does it need to be after fragment shading?

63

Depth Test



64

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

Depth Test

65

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

Blending

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

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

66

Blending

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

- Example: Translucency
- Additive Blending

C_{dest} = (C_{source}.rgb)(C_{source}.a) + (C_{dest}.rgb);

- Alpha Blending

C_{dest} = (C_{source}.rgb)(C_{source}.a) +
(C_{dest}.rgb) (1 - C_{source}.a);

67

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

Graphics Pipeline Walkthrough

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

- After all that, write to the framebuffer!

Output image buffer

68

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

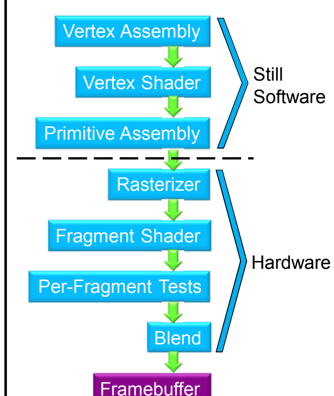
- Interactive software rendering (no GPUs yet)
- NOTE: SGI was building interactive rendering supercomputers, but this was beginning of interactive 3D graphics on PC

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, MAD, sqrt, etc.

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/

73

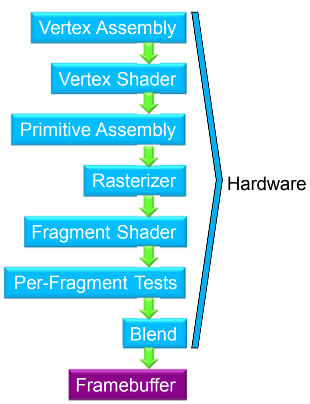
Aside: Mario Kart Wii

- High fragment load / low vertex load?

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

74

NVIDIA GeForce 256 (1999)



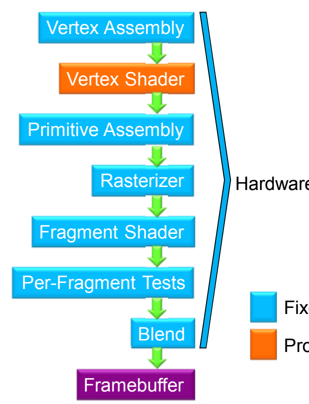
In hardware:

- 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

75

NVIDIA GeForce 3 (2001)



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

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

Fixed-function stage

Programmable stage

76

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

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

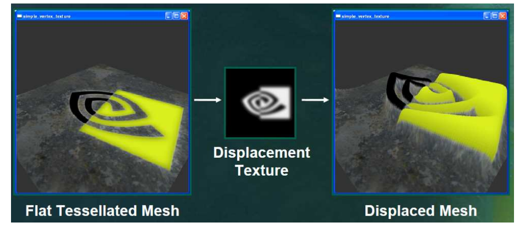


Image from ftp://download.nvidia.com/developer/presentations/2004/GPU_Jackpot/Shader_Model_3.pdf

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

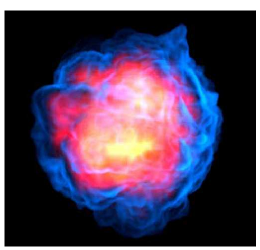


Image from ftp://download.nvidia.com/developer/presentations/2004/GPU_Jackpot/Shader_Model_3.pdf

Dynamic Branches

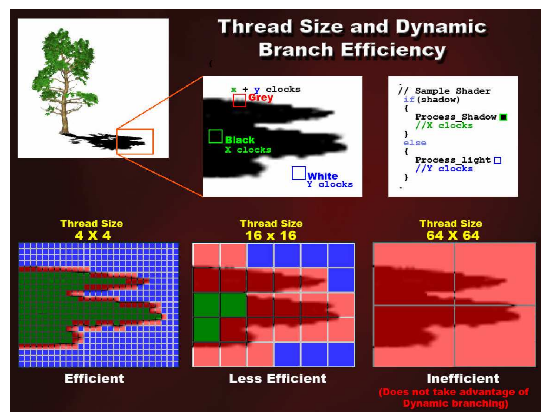


Image from http://developer.amd.com/media/gpu_assets/03_Clever_Shader_Tricks.pdf

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)

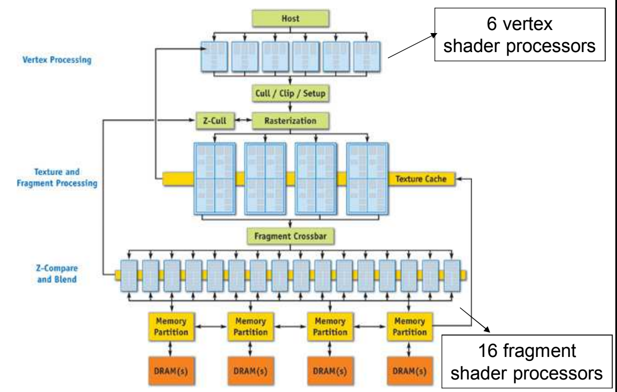
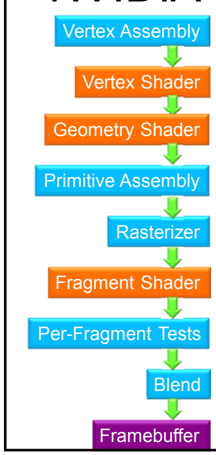


Image from http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter30.html

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

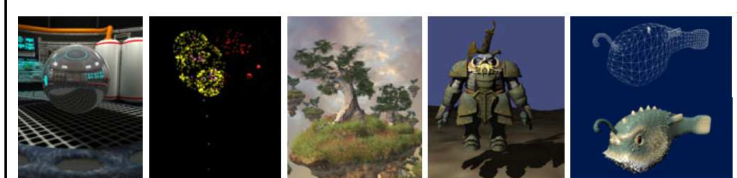
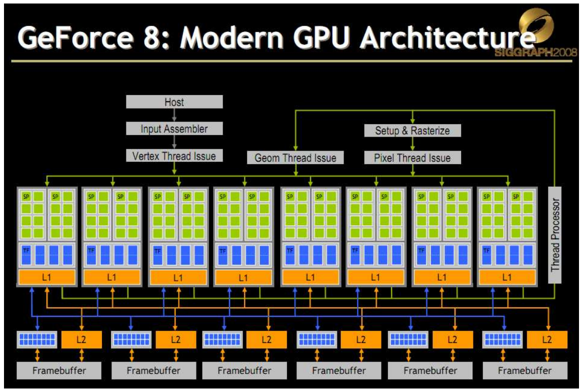


Figure 5: From left — render to cube map, particle system, instancing, shadow volume, displacement mapping.

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

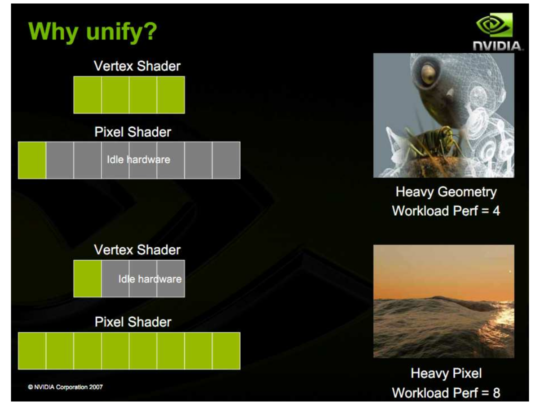
NVIDIA G80 Architecture



85

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

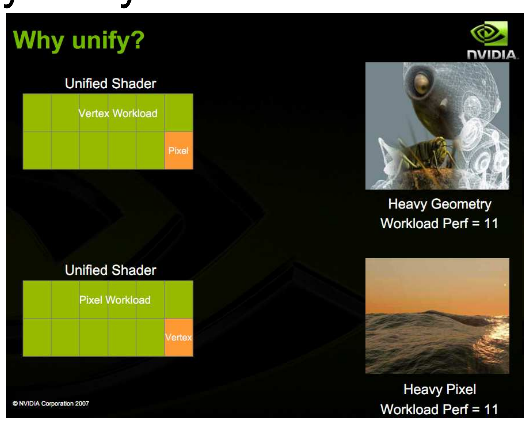
Why Unify Shader Processors?



86

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

Why Unify Shader Processors?



87

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

Terminology

Shader Model	Direct3D	OpenGL	Video card Example
3	9	2.x	NVIDIA GeForce 6800 ATI Radeon X800
4	10.x	3.x	NVIDIA GeForce 8800 ATI Radeon HD 2900
5	11.x	4.x	NVIDIA GeForce GTX 480 ATI Radeon HD 5870

88

Shader Capabilities

	SM 2.0/2.X	SM 3.0	SM 4.0
Introduced	DX 9.0, 2002	DX 9.0c, 2004	DX 10, 2007
VS Instruction Slots	256	$\geq$ 512[a]	4096
VS Max. Steps Executed	65536	65536	∞
PS Instruction Slots	$\geq$ 96[b]	$\geq$ 512[a]	$\geq$ 65536[a]
PS Max. Steps Executed	$\geq$ 96[b]	65536	∞
Temp. Registers	$\geq$ 12[a]	32	4096
VS Constant Registers	$\geq$ 256[a]	$\geq$ 256[a]	14×4096[c]
PS Constant Registers	32	224	14×4096[c]
Flow Control, Predication	Optional[d]	Yes	Yes
VS Textures	None	4[e]	128×512[f]
PS Textures	16	16	128×512[f]
Integer Support	No	No	Yes
VS Input Registers	16	16	16
Interpolator Registers	8[g]	10	16/32[h]
PS Output Registers	4	4	8

89

Table courtesy of A K Peters, Ltd. http://www.realtimerendering.com/

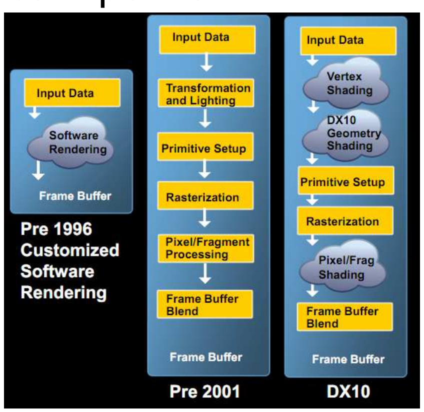
Shader Capabilities

	SM 2.0/2.X	SM 3.0	SM 4.0
Introduced	DX 9.0, 2002	DX 9.0c, 2004	DX 10, 2007
VS Instruction Slots	256	$\geq$ 512[a]	4096
VS Max. Steps Executed	65536	65536	∞
PS Instruction Slots	$\geq$ 96[b]	$\geq$ 512[a]	$\geq$ 65536[a]
PS Max. Steps Executed	$\geq$ 96[b]	65536	∞
Temp. Registers	$\geq$ 12[a]	32	4096
VS Constant Registers	$\geq$ 256[a]	$\geq$ 256[a]	14×4096[c]
PS Constant Registers	32	224	14×4096[c]
Flow Control, Predication	Optional[d]	Yes	Yes
VS Textures	None	4[e]	128×512[f]
PS Textures	16	16	128×512[f]
Integer Support	No	No	Yes
VS Input Registers	16	16	16
Interpolator Registers	8[g]	10	16/32[h]
PS Output Registers	4	4	8

90

Table courtesy of A K Peters, Ltd. http://www.realtimerendering.com/

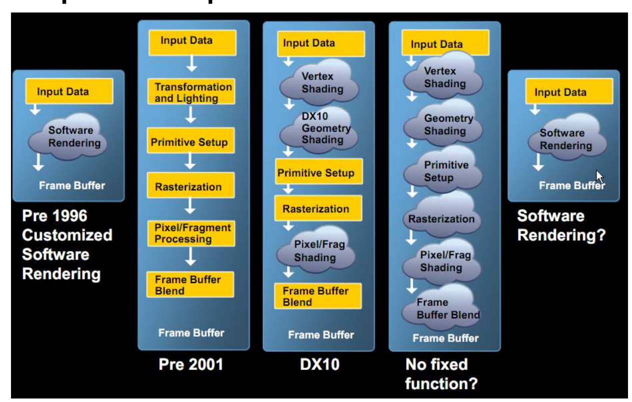
Evolution of the Programmable Graphics Pipeline



91

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

Evolution of the Programmable Graphics Pipeline



92

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