Introduction to Computer Graphics

6. GPU and Shaders

I-Chen Lin
National Chiao Tung University

Textbook: E.Angel, D. Shreiner Interactive Computer Graphics, 6th Ed., Pearson
Ref: D.D. Hearn, M. P. Baker, W. Carithers, Computer Graphics with OpenGL, 4th Ed., Pearson

The Development of Graphics Cards (consumer-level): Early 90’s

no hardware support

- VGA cards in the early 90’s 演算法在CPU裡算

  把mem某一個影像或texture送到monitor上

  - Just output designated “bitmap”.
  - Some with 2D acceleration, ex. “Bitblt”
  - Ex. S3 早期的紅色
- Interactive 3D(or 2.5D) games relied on software rendering. 計算視角
  - There were hardware graphics pipelines on workstations, e.g. SGI.

Figures from https://en.wikipedia.org/wiki/Wolfenstein_3D

The Development of Graphics Cards (consumer-level): Late 90's

- 3D accelerators (90's)
  - Fixed-function pipelines. hw1内容
  - E.g. S3, Voodoo, Nvidia, ATI, 3D Labs….
  - Some of them had to work with a standard VGA card.

顯示卡

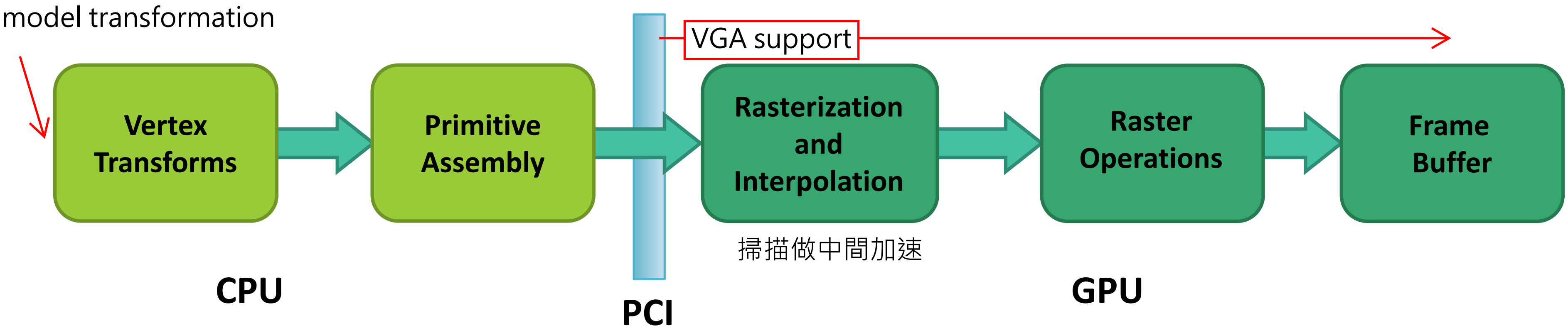
3Dfx Voodoo (1996)

很多多邊形把角色拼起來=>三角形shading重要：
gouraud/smooth shading是OpenGL default

- One of the first true 3D game cards
- Worked by supplementing a standard 2D video card.
- Did not do vertex transformations (they were evaluated in the CPU)
- Did texture mapping, z-buffering.

VGA卡、圖形加速卡串接

en.wikipedia.org/wiki/3dfx_Interactive



Modified from S. Venkatasubramanian and J. Kider, “Evolution of the Programmable Graphics Pipeline”

G : graphics ATI->AMD

GeForce/Radeon 7500 (1998)

fixed pipeline，送到卡上做內插

- Main innovation: shifting the transformation and lighting calculations to the GPU
- Allowed multi-texturing: giving bump maps, light maps, and others.

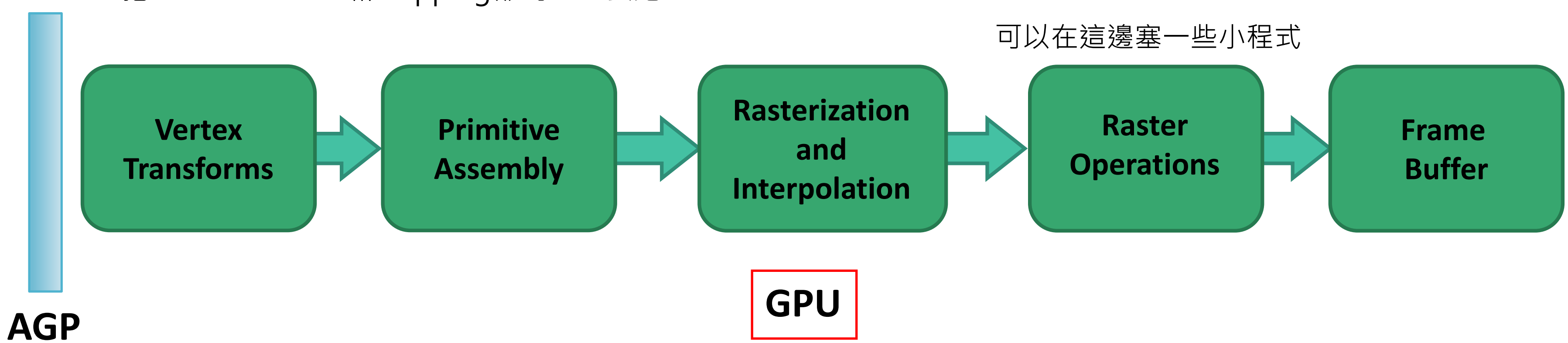
translation/lighting

- Faster AGP bus instead of PCI

把transformation和mapping都到GPU去跑

en.wikipedia.org/wiki/GeForce_256

可以在這邊塞一些小程式



Modified from S. Venkatasubramanian and J. Kider, “Evolution of the Programmable Graphics Pipeline”

The Development of Graphics Cards (consumer-level): after 2001

可以塞程式

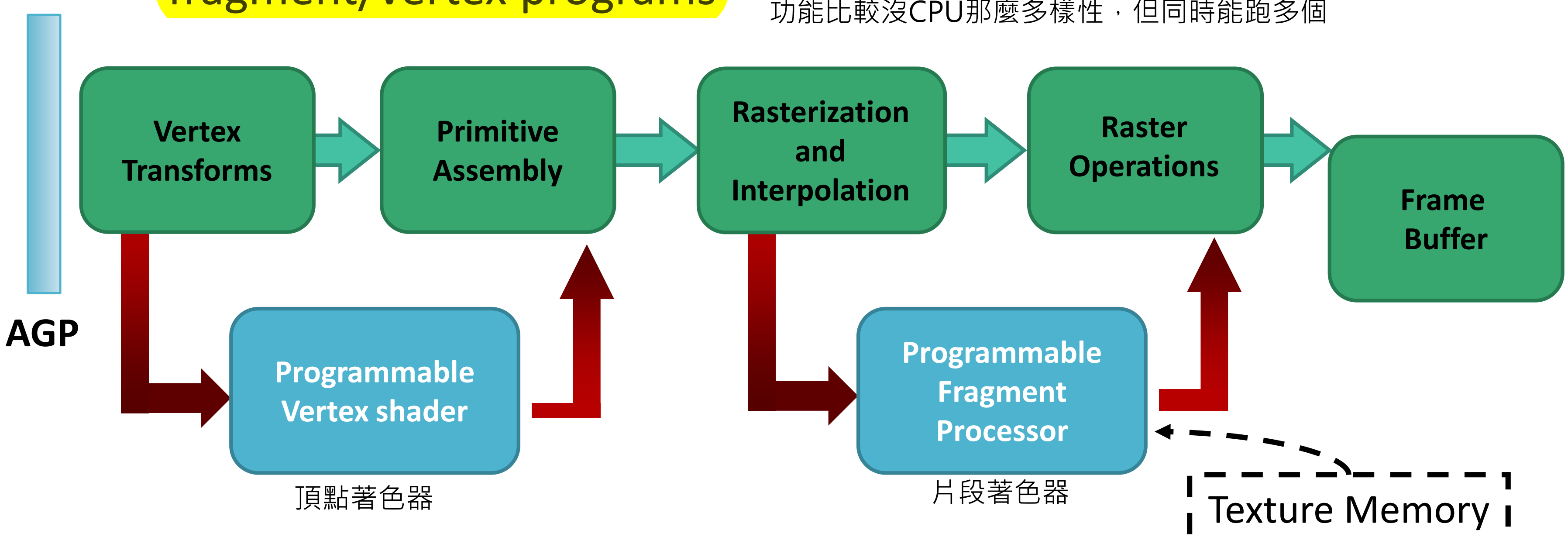
- Programmable pipelines on GPU
- GeForce3/Radeon 8500(2001)
  - Programmable vertex computations: up to 128 instructions
  - Limited programmable fragment computations: 8-16 instructions

https://en.wikipedia.org/wiki/GeForce_3_series

The Development of Graphics Cards (consumer-level): after 2001 (cont.)

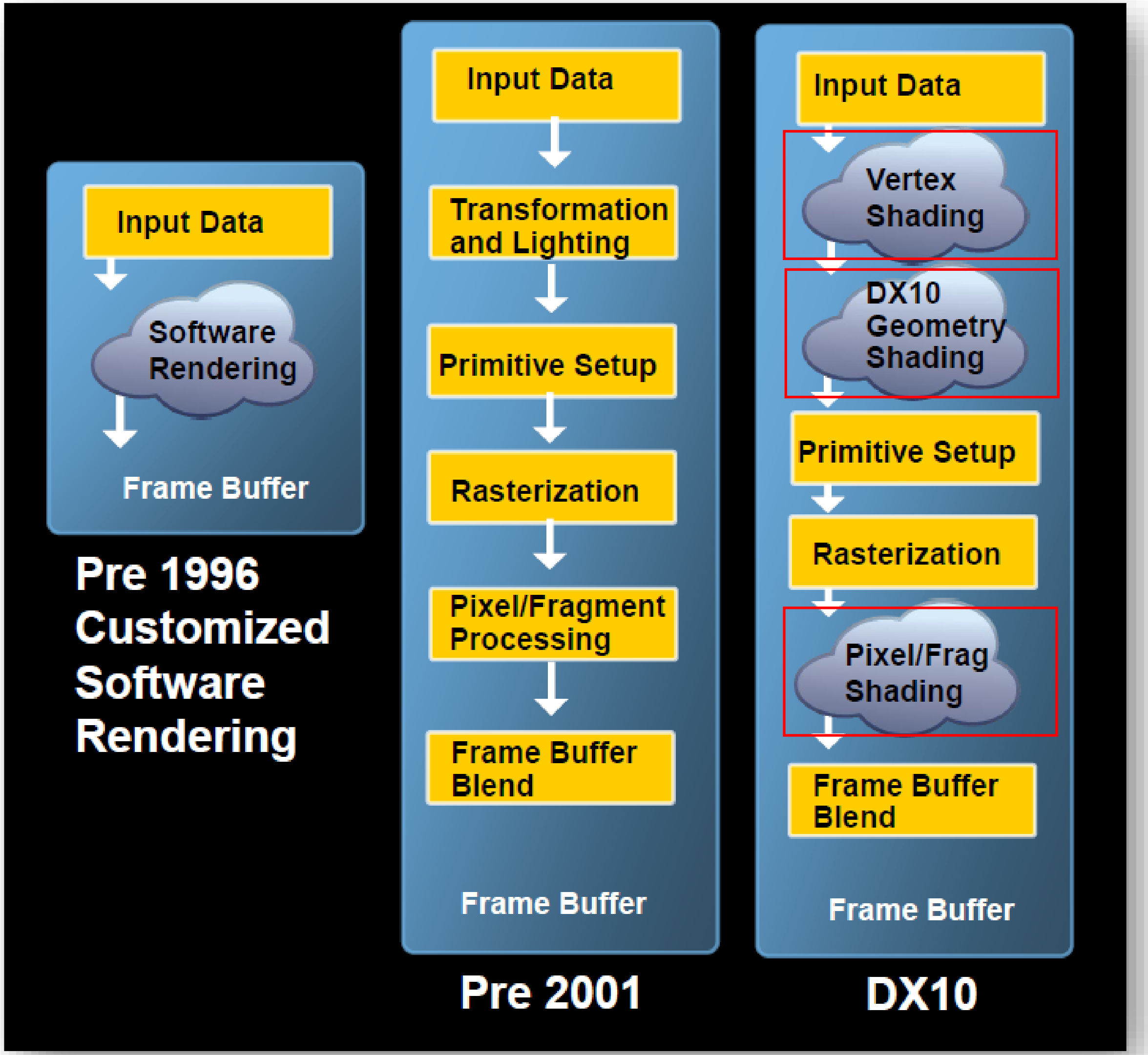
- Radeon 9700/GeForce FX (2002)
  - the first generation of fully-programmable graphics cards 可以塞程式
  - Different versions have different resource limits on fragment/vertex programs

GPU thread都是跑同一個function：
功能比較沒CPU那麼多樣性，但同時能跑多個



Modified from S. Venkatasubramanian and J. Kider, "Evolution of the Programmable Graphics Pipeline"

Evaluation of Graphics Pipeline



…………

Figure from: M. Houston, "Beyond Programmable Shading Retrospective" slides

GPU & Shaders : the new age of real-time graphics

- Programmable pipelines.
- Supported by high-end commodity cards
  - NVIDIA, AMD/ATI, etc.

en.wikipedia.org/wiki/GeForce_10_series

www.amd.com/zh-hant/products/graphics/radeon-rx-570

Why is It So Remarkable?

- We can do lots of cool stuff in real-time, without overworking the CPU. =>GPU大量平行運算
  - Phong Shading 每個頂點用小程式計算=>得到頂點顏色
  - Bump Mapping 圖形鼓起來
  - Particle Systems EX : 火焰
  - Animation
  - ……

支援矩陣運算

- Beyond real-time graphics: GP-GPU, e.g. CUDA, OpenCL (Open Computing Language)
  - Scientific Data Processing
  - Computer vision
  - Deep learning
  - ……

Programmable Components

- **Shader**: programmable processors. shader language
  - Replacing fixed-function vertex and fragment processing, and so forth.
- Types of shaders:
  - 必備 **Vertex shaders** 每個點都會執行function(平行運算)，但只能拿的到自己頂點的資料
    - Dealing with per-vertex functions.
    - We can control the lighting and position of each vertex.
  - 必備 **Fragment shaders** 在螢幕上的位置確定
    - Dealing with per-pixel functions.
    - We can control the color of each pixel by user-defined programs.

  - Geometry shaders (DirectX 10, SM 4+)
  - New shaders (hull, domain) in DirectX11, SM5

Programmable Components (cont.)

- Software Support
  - Direct X 8 , 9, 10, 11, 12, …
  - OpenGL Extensions
  - OpenGL Shading Language (GLSL)
  - OpenGL for Embedded Systems (OpenGL ES)
  - Cg (*C for Graphics*)
  - Metal Shading Language (by Apple)
  - ………….

pixel : 最後畫在畫面上的
fragment : 會有多個fragment在pixel上

input point=>output point(進出都是position)

Essential GLSL pipeline (Vert.+Frag. Shaders)

可以用小程式控制光和顏色，
但位置已經固定了

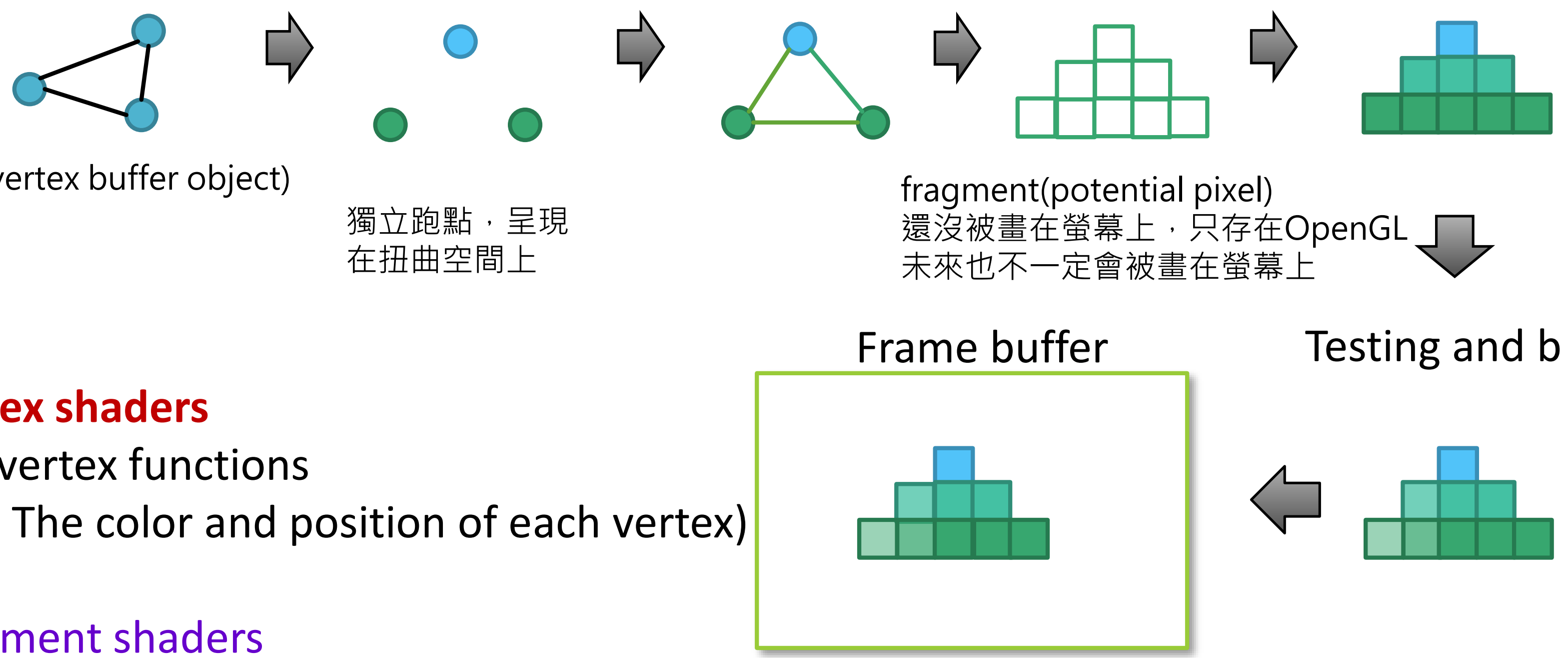
形變or繼承矩陣

點和邊都是primitive

position、color...

Input data | Vertex Shader | Primitives | Rasterization | Fragment Shader

近似於pixel



VBO(vertex buffer object)

獨立跑點，呈現
在扭曲空間上

fragment(potential pixel)
還沒被畫在螢幕上，只存在OpenGL
未來也不一定會被畫在螢幕上

Frame buffer

Testing and blending

Vertex shaders
per-vertex functions
(E.g. The color and position of each vertex)

Fragment shaders
per-fragment (pixel) function.
(E.g. The color of each fragment)

go to vb2 p.14

Vertex Shaders

- Per-vertex calculations performed here
  - Without knowledge about other vertices (parallelism)

  - Your program take responsibility for:
    - Vertex transformation
    - Normal transformation
    - (Per-Vertex) Lighting
    - Color material application and color clamping
    - Texture coordinate generation

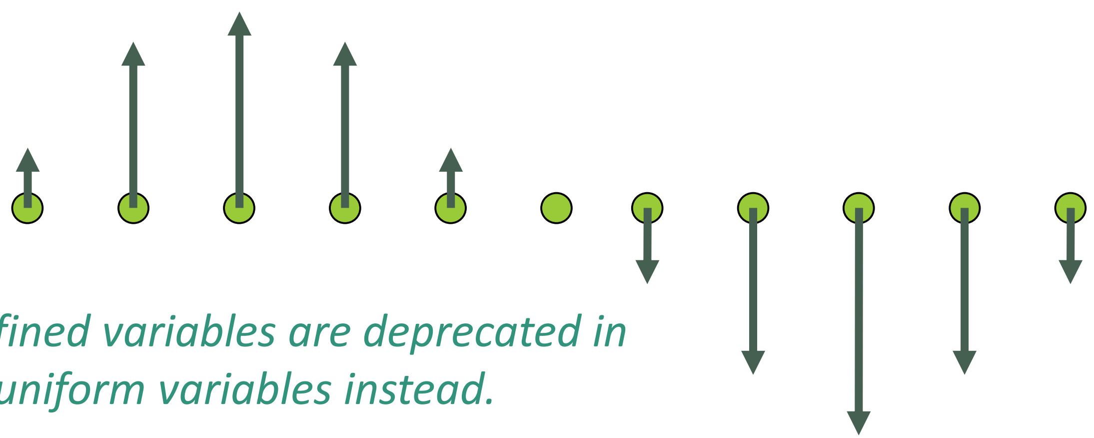
Vertex Shader Applications

- We can control movement with uniform variables and vertex attributes
  - Time
  - Velocity
  - Gravity
- Moving vertices
  - Morphing
  - Wave motion
  - ……
- Lighting
  - More realistic models
  - Cartoon shaders

Applications: Wave Motion Vertex Shader

Uniform: passing parameters to vertex and fragment shaders.

```
……………………………………
uniform float time;
uniform float xs, zs;
void main()
{
float s;
s = 1.0 + 0.1*sin(xs*time)*sin(zs*time);
gl_Vertex.y = s*gl_Vertex.y;
gl_Position =
gl_ModelViewProjectionMatrix*gl_Vertex;
}
```

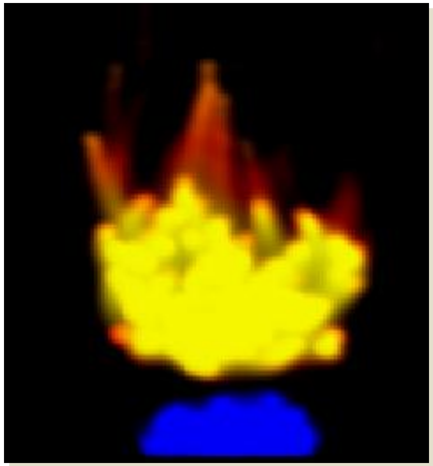


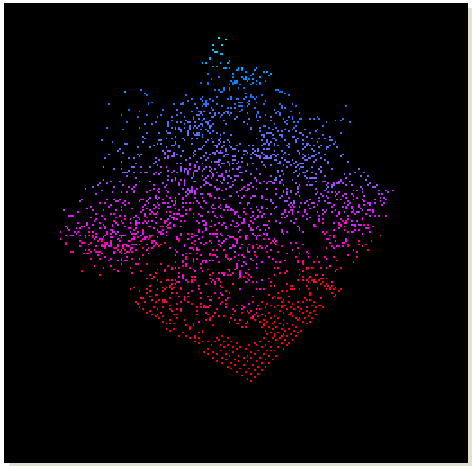
Note: Several gl_ predefined variables are deprecated in the newer version. Use uniform variables instead.

Applications: Particle Systems

Uniform: passing parameters to vertex and fragment shaders.

```
................................
uniform vec3  init_vel;
uniform float g, m, t;
void main()
{
vec3 object_pos;
object_pos.x = gl_Vertex.x + vel.x*t;
object_pos.y = gl_Vertex.y + vel.y*t
+ g/(2.0*m)*t*t;
object_pos.z = gl_Vertex.z + vel.z*t;
gl_Position =
gl_ModelViewProjectionMatrix*
vec4(object_pos,1);
}
```





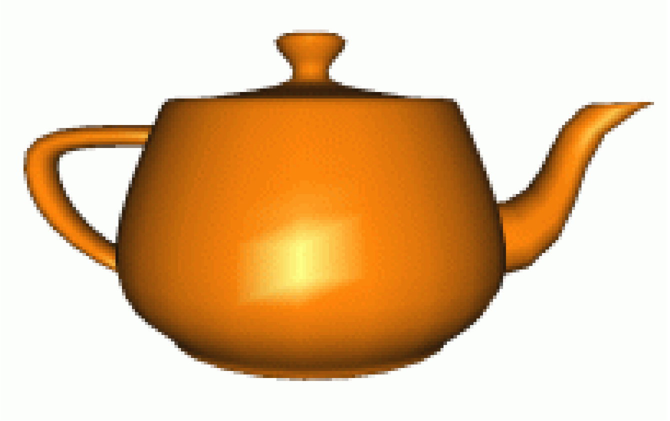
Note: Several gl_ predefined variables are deprecated in the newer version. Use uniform variables instead.

Fragment Shaders

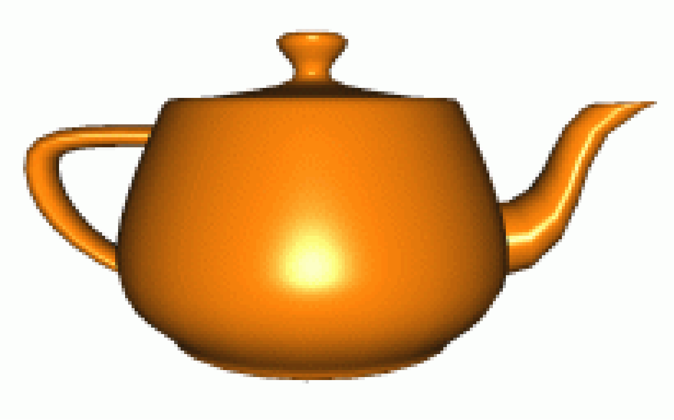
- What is a fragment?
  - Cg Tutorial says: “You can think of a fragment as a ‘potential pixel’”
- Perform per-pixel calculations
  - Without knowledge about other fragments (parallelism)
- Your program’s responsibilities:
  - Operations on interpolated values
  - Texture access and application
  - Other functions: fog, color lookup, etc.

Fragment Shader Applications

(Per-pixel) Phong shading



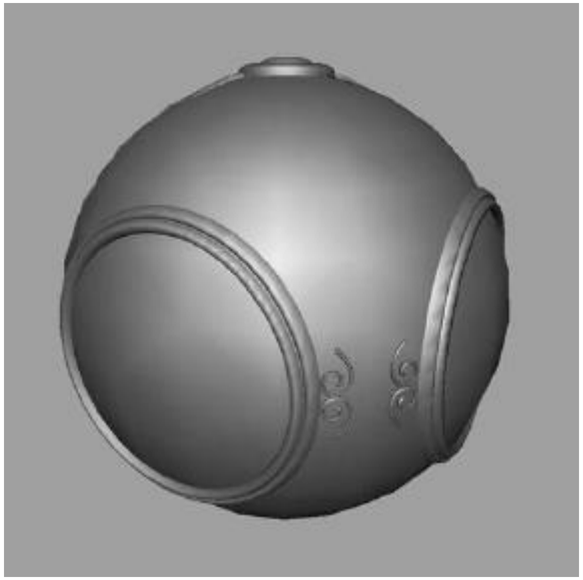
Per-vertex lighting



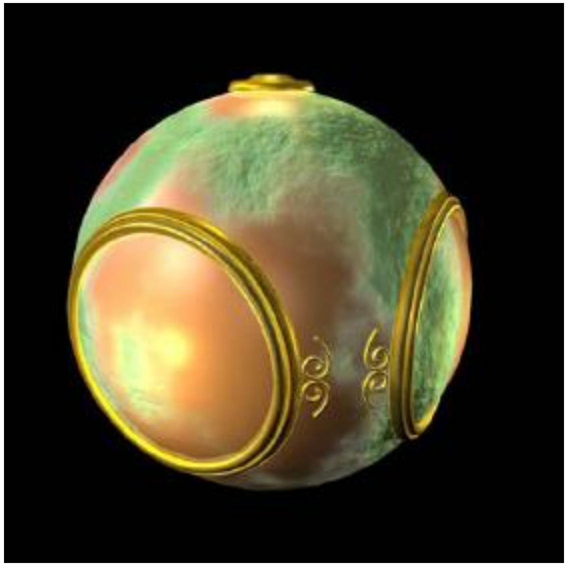
Per-fragment lighting

Figures from http://www.lighthouse3d.com/opengl/glsl/

Fragment Shader Applications



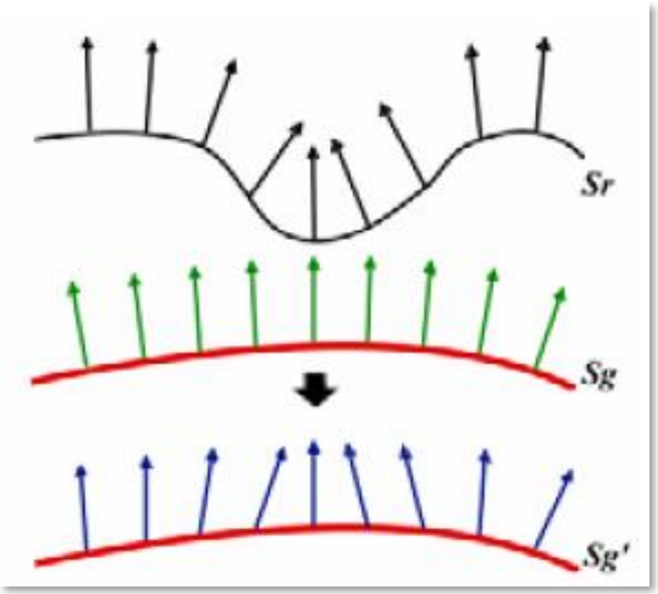
smooth shading

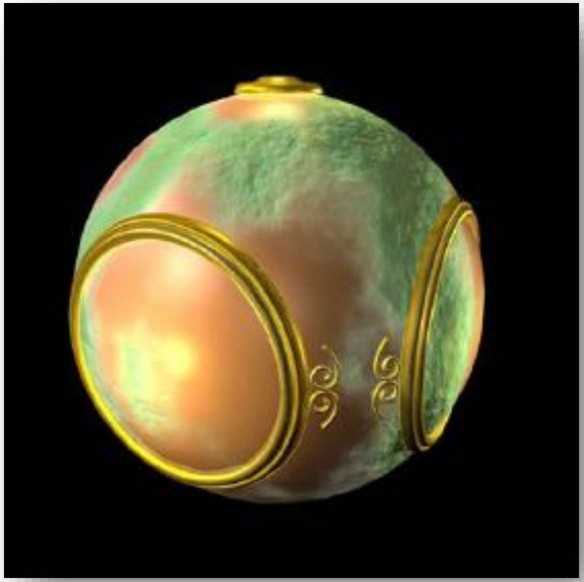


bump mapping

Bump Mapping

- Perturb normal for each fragment
- Store perturbation as textures





Toon Shading

ftransform(): result from the GL fixed-function transformation pipeline

Note: ***varying****, communicating between vertex and fragment. Use* ***in out*** *variables in newer versions.*

- The vertex shader then becomes:

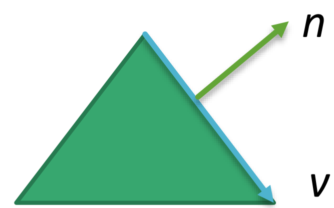
```
………………………
out vec3 vnormal;
void main() {
    vnormal = gl_NormalMatrix * gl_Normal;
    gl_Position = ftransform(); }
```

- The fragment shader becomes

```
………………………
in vec3 vnormal;
void main() {
    float intensity; vec4 color;
    vec3 n = normalize(vnormal);
    intensity = dot(vec3(gl_LightSource[0].position),n);
    if (intensity > 0.95) color = vec4(1.0,0.5,0.5,1.0);
    else if (intensity > 0.5) color = vec4(0.6,0.3,0.3,1.0);
    else if (intensity > 0.25) color = vec4(0.4,0.2,0.2,1.0);
    else color = vec4(0.2,0.1,0.1,1.0);
    gl_FragColor = color; }
```

Example from http://www.lighthouse3d.com/opengl/glsl/

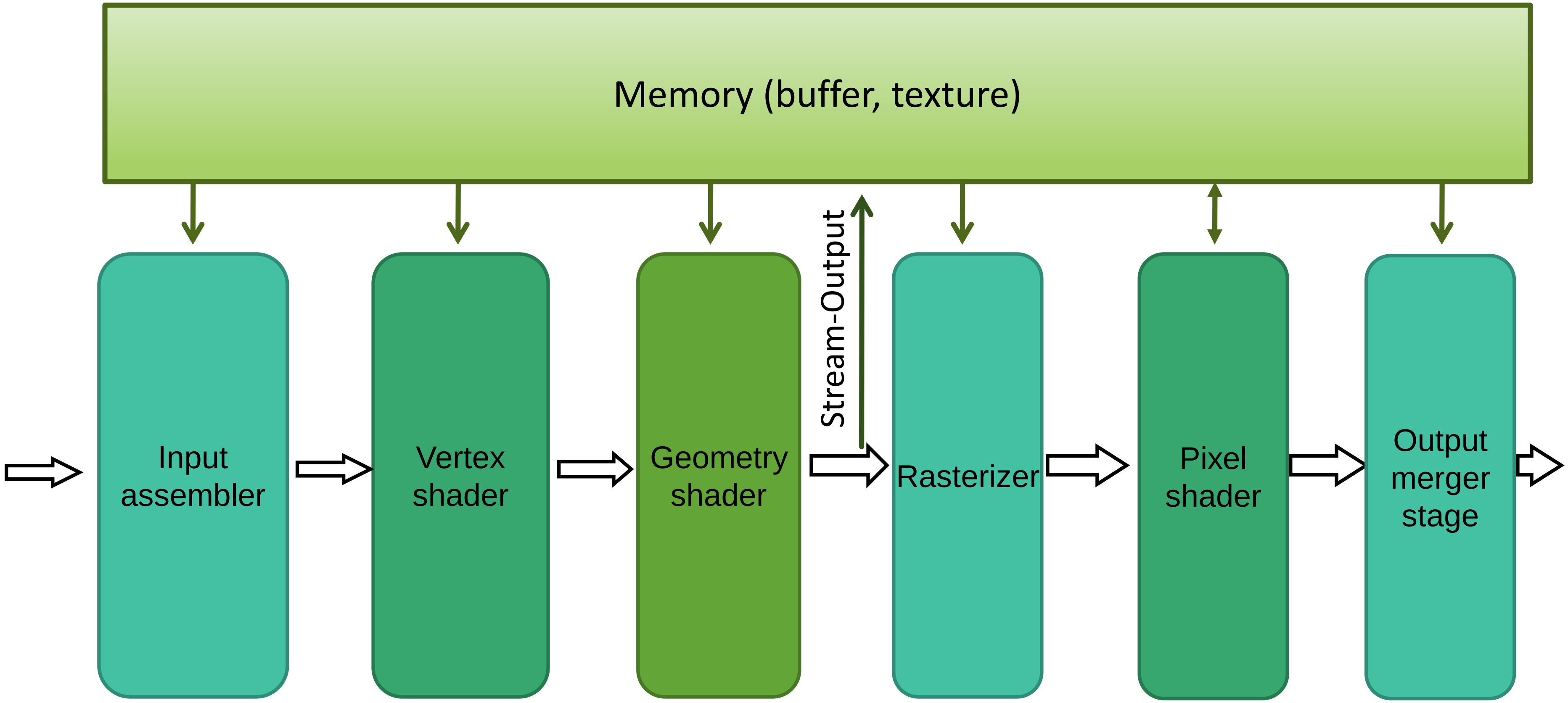
gl_NormalMatrix



$n \cdot v = 0$

- Can we directly apply the modelview matrix M to a normal vector ?
  - Problem: If the upper-left 3x3 submatrix M_s is not orthogonal, $n' = M_s n$ is not perpendicular to $v' = M_s v$

With the Geometry Shader



Direct3D 10 pipeline stage from MSDN of Microsoft

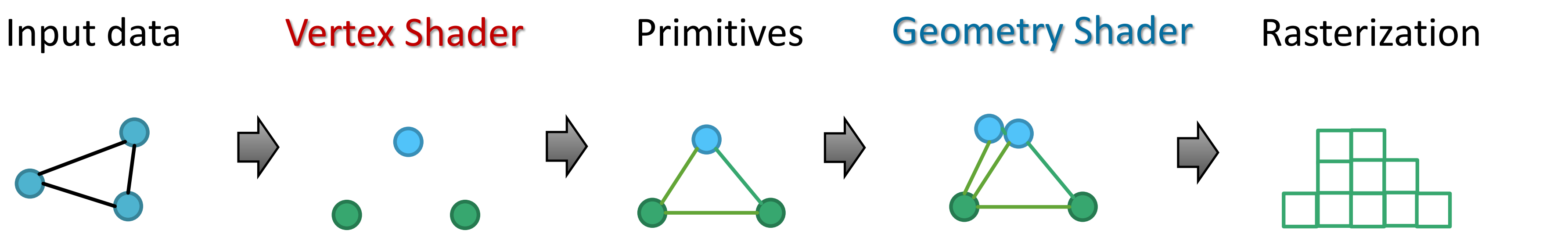
D3D 10 Pipeline

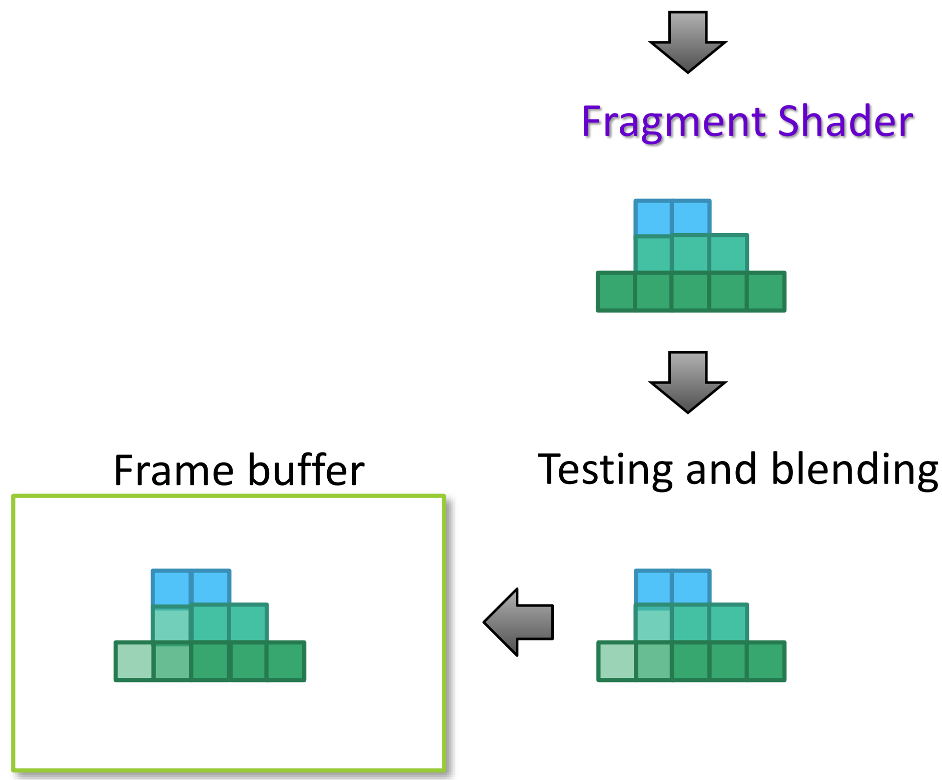
- **Input assembler**: supplies data (triangles, lines and points) to the pipeline.
- **Vertex shader**: processes vertices, such as transformations, skinning, and lighting.
- **Geometry shader**: processes entire primitives.
  - 3 vertices: a triangle, 2 vertices: a line, or 1 vertex: a point.
  - The Geometry shader supports limited geometry amplification and de-amplification. (discard the primitive, or emit one or more new primitives)
  - E.g. Subdivision, point ->billboard, silhouette edge -> fur, etc.
- **Stream-output stage**:
  - Data can be streamed out and/or passed into the rasterizer. Data streamed out to memory can be recirculated back into the pipeline as input data or read-back from the CPU.

D3D 10 Pipeline (cont.)

- **Rasterizer**: clips primitives, prepares primitives for the pixel shader and determines how to invoke pixel shaders.
- **Pixel shader**: receives interpolated data for a primitive and generates per-pixel data, such as color.
- **Output-merger stage**:
  - combines various types of output data (pixel shader values, depth and stencil information) with the contents of the render target and depth/stencil buffers to generate the final pipeline result.

GLSL pipeline (Vert.+Geo.+Frag. Shaders)



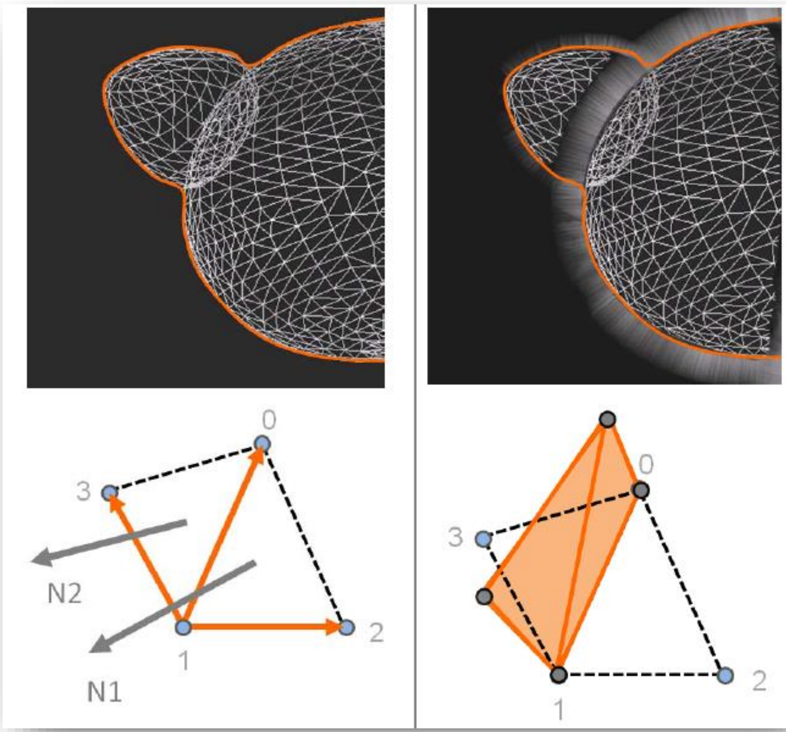


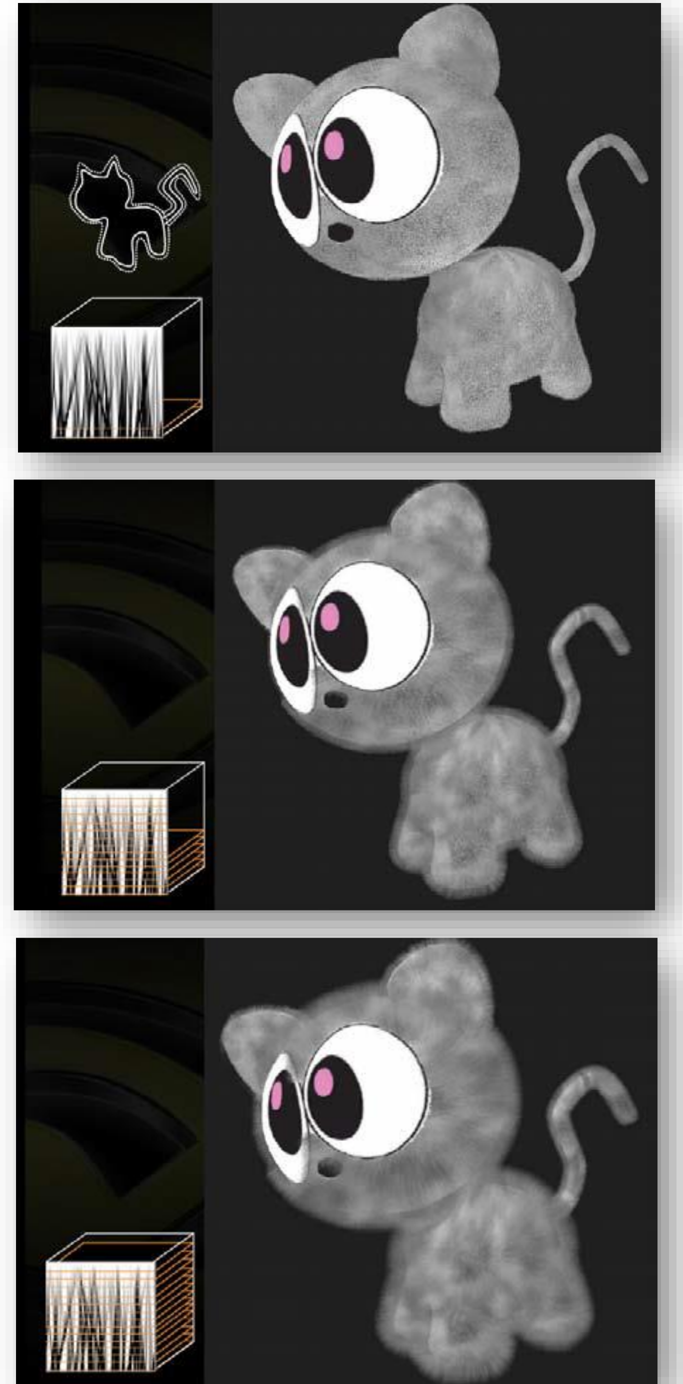
Vertex shaders
per-vertex functions

Geometry shaders
Primitive processing
(E.g. transformation, generating zero to multiple primitives)

Fragment shaders
per-fragment (pixel) function.

D3D 10 Pipeline (cont.)





Figures from NVIDIA DirectX10 SDK Doc:
Fur (using Shells and Fins)

D3D 11 Pipeline

- In D3D10, the Geometry shader may subdivide the surfaces by multiple passes.
- D3D11 improves the tessellation ability by three new stages: hull shader, tessellator, domain shader.
- The tessellated patches can still be applied to geometry shaders. E.g. point ->billboard, silhouette edge -> fur, etc.

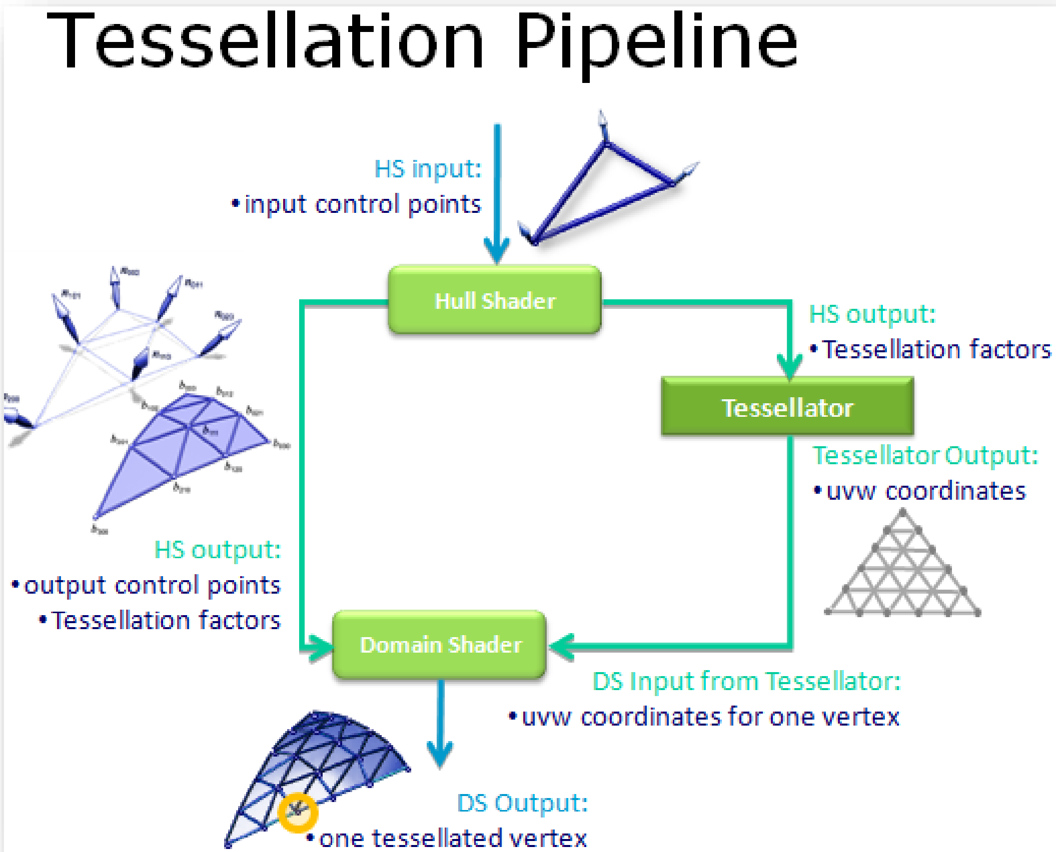
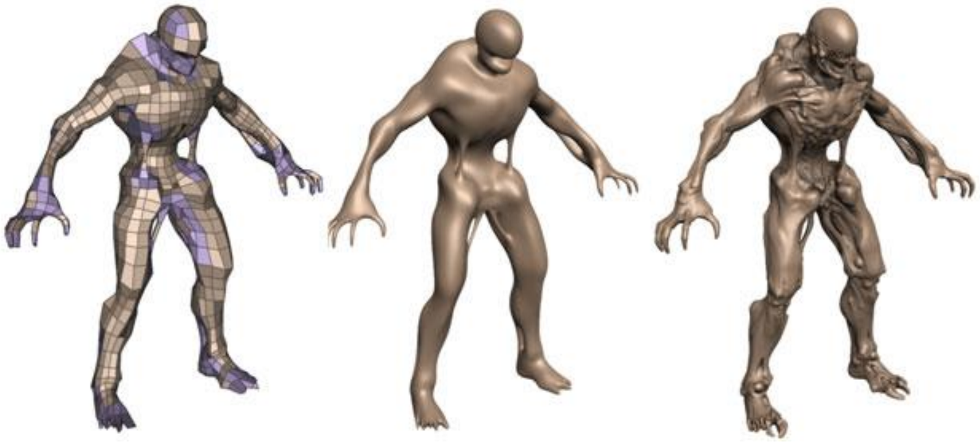


Figure from:
developer.download.nvidia.com/presentations/2009/GDC/GDC09_D3D11
Tessellation.pdf

Figure from: vulkan-tutorial.com/Drawing_a_triangle/Graphics_pipeline_basics/Introduction

D3D 11 Tessellation

Model refinement



Tessellation with displacement mapping

Figures from : https://www.nvidia.com.tw/object/tessellation_tw.html

End of Chapter 6