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

- VGA cards in the early 90's
  - Just output designated "bitmap".
  - Some with 2D acceleration, ex. "Bitblt"
  - Ex. S3
- Interactive 3D(or 2.5D) games relied on software rendering.
  - There were hardware graphics pipelines on workstations, e.g. SGI.

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

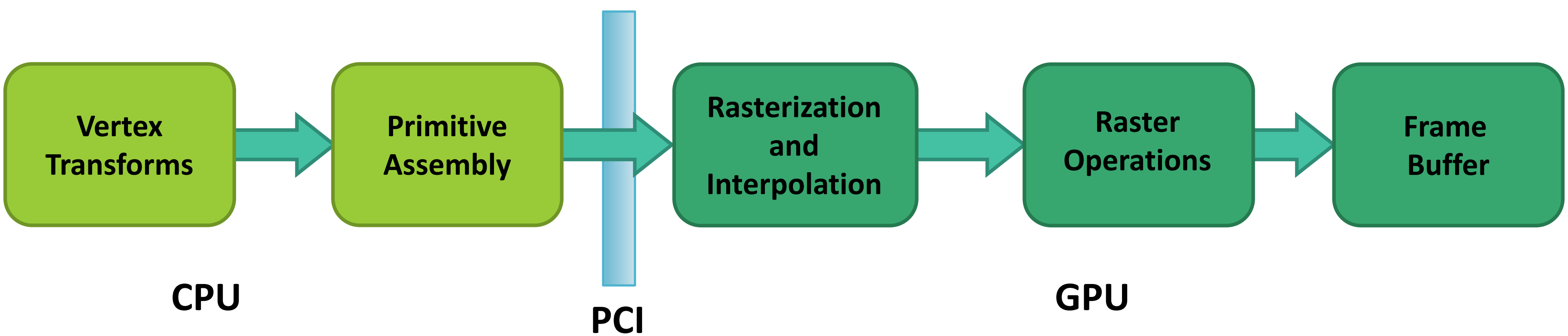
The Development of Graphics Cards (consumer-level): Late 90’s

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

3Dfx Voodoo (1996)

- 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.

en.wikipedia.org/wiki/3dfx_Interactive

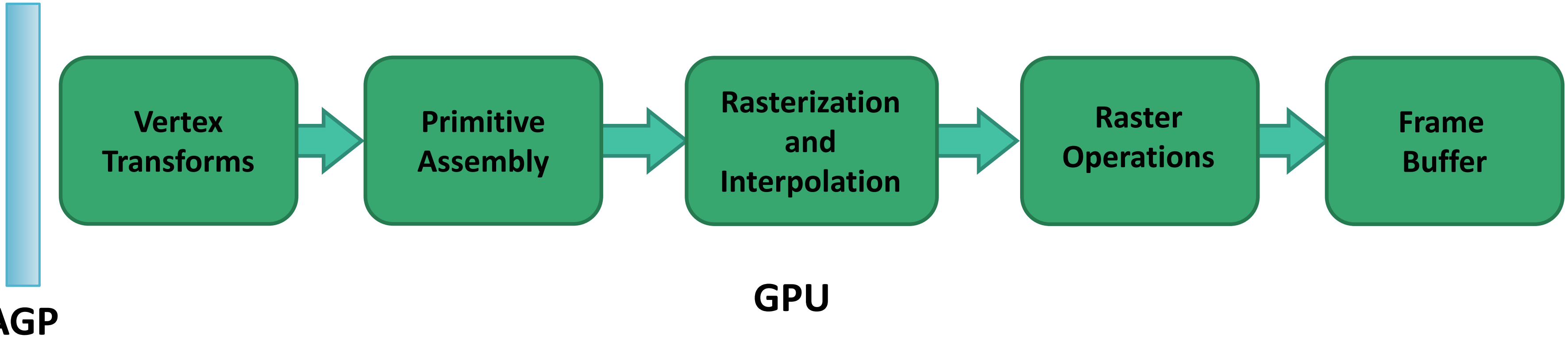


Modified from S. Venkatasubramanian and J. Kider, “Evolution of the Programmable Graphics Pipeline”

GeForce/Radeon 7500 (1998)

- Main innovation: shifting the transformation and lighting calculations to the GPU
- Allowed multi-texturing: giving bump maps, light maps, and others.
- Faster AGP bus instead of PCI

en.wikipedia.org/wiki/GeForce_256



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

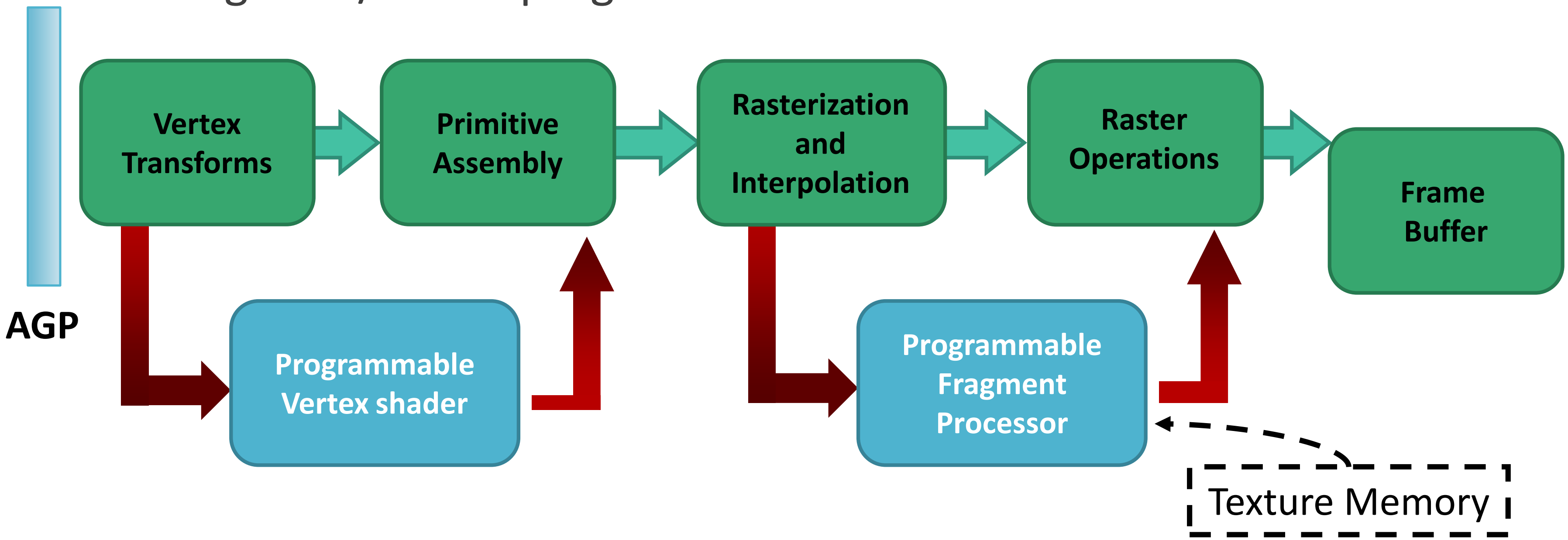
The Development of Graphics Cards (consumer-level): after 2001

- 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

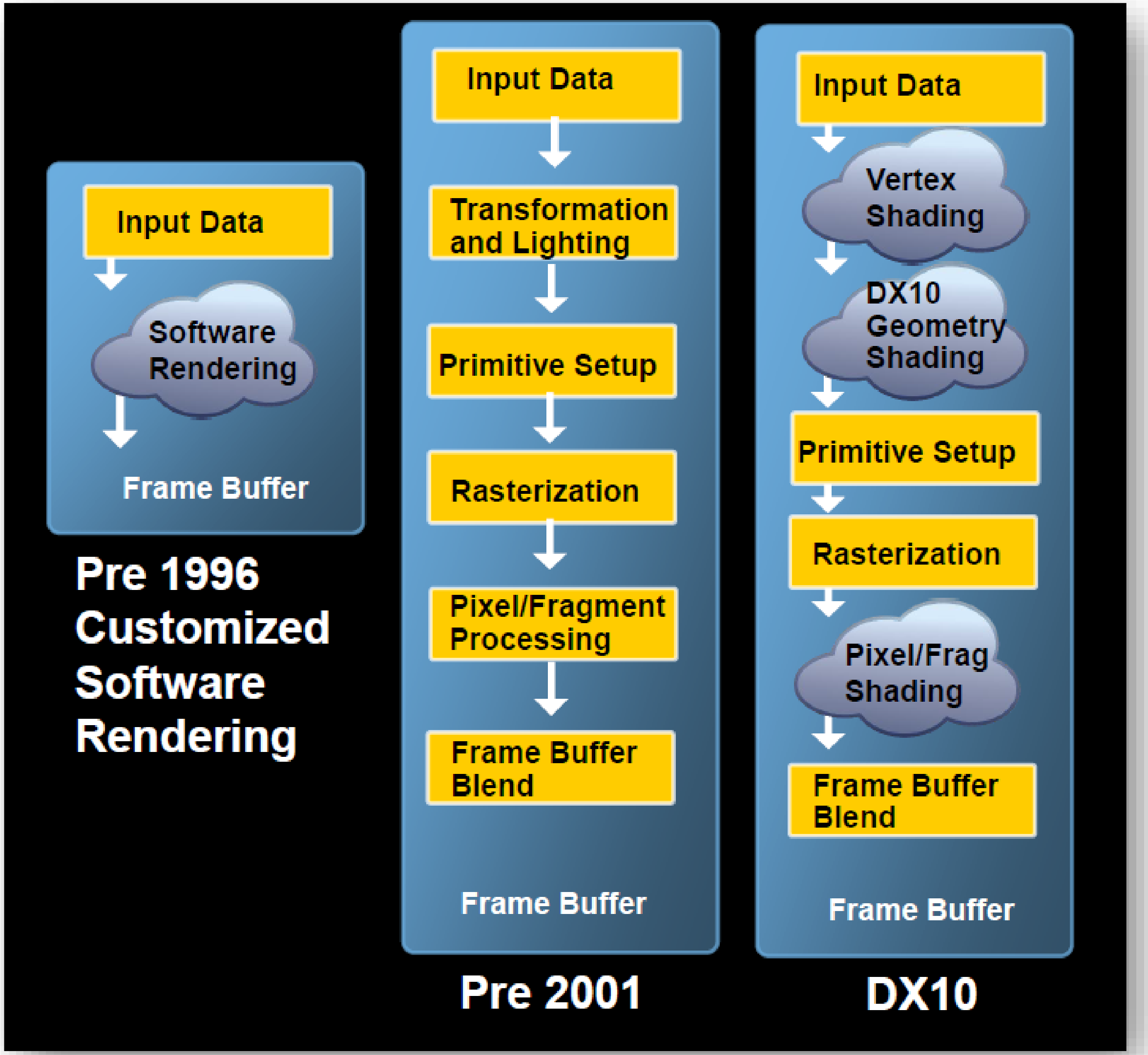
The Development of Graphics Cards (consumer-level): after 2001 (cont.)

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



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.
  - Phong Shading
  - Bump Mapping
  - Particle Systems
  - Animation
  - ……
- Beyond real-time graphics: GP-GPU, e.g. CUDA, OpenCL (Open Computing Language)
  - Scientific Data Processing
  - Computer vision
  - Deep learning
  - ……

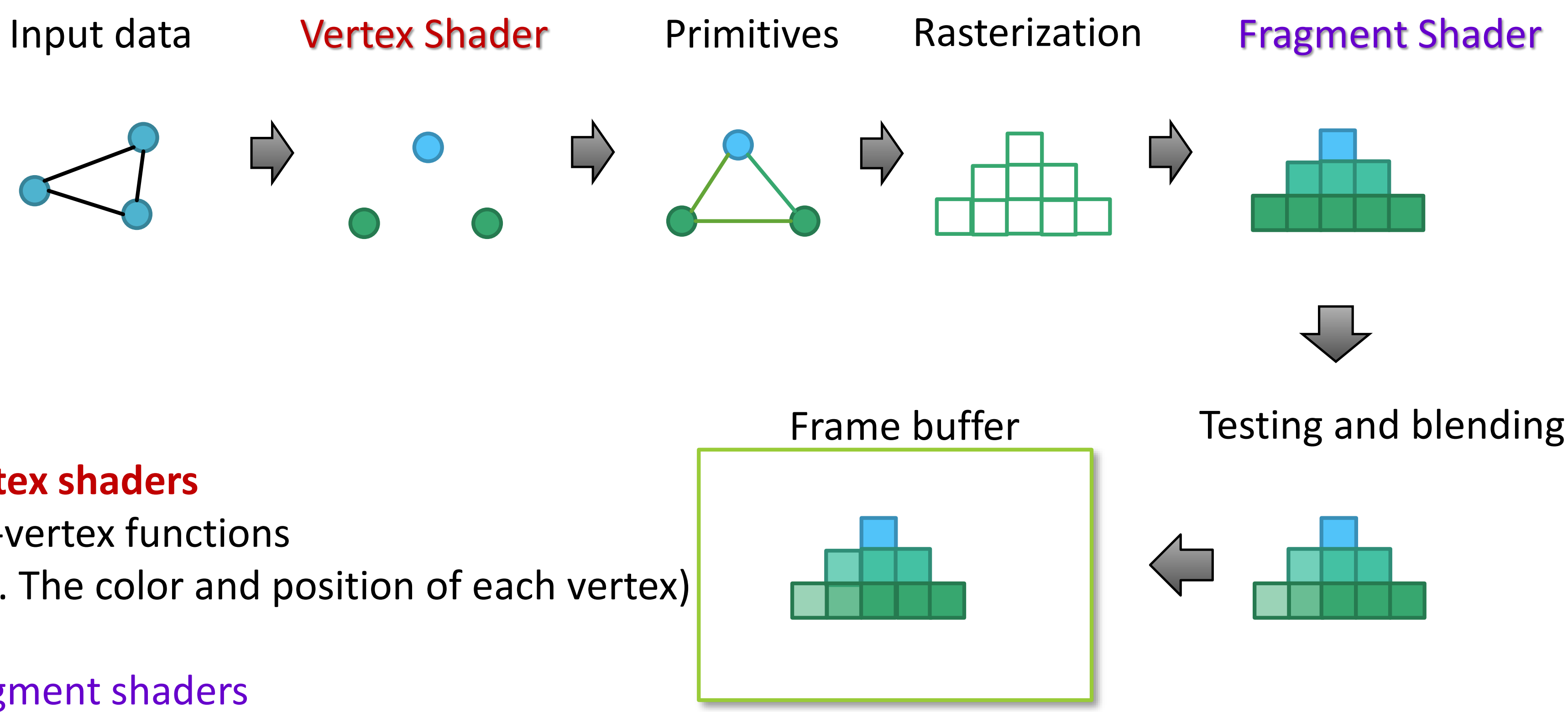
Programmable Components

- **Shader**: programmable processors.
  - Replacing fixed-function vertex and fragment processing, and so forth.
- Types of shaders:
  - **Vertex shaders**
    - Dealing with per-vertex functions.
    - We can control the lighting and position of each vertex.
  - Fragment shaders
    - 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)
  - ………….

Essential GLSL pipeline (Vert.+Frag. Shaders)



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)

What about GLSL programs?

- Besides your main program (e.g. main.cpp), there are additional shader codes.

- These code can be character strings in your .cpp, but we usually put them in separate files (e.g. ooo.vs or ooo.vert, xxx.fs or xxx.frag).

- In a GLSL program, you can use multiple (different) shader codes to demonstrate different illumination algorithms for objects or regions. 一個shader不是搭配一個function，可用不同shader對應同一物體

*不能用glBegin、glEnd找點：效率較差

A simple example of shader codes

Vert buffer passed from main.cpp

VAO
VBO

```
float vertices[] = {
   0.5f, -0.5f, 0.0f,
  -0.5f, -0.5f, 0.0f,
   0.0f,  0.5f, 0.0f
};
```

input是VBO，裡面會存個array

Vertex shader code

宣告版本：330核心版

```
#version 330 core
layout (location = 0) in vec3 aPos; // the position variable has attribute position 0

out vec4 vertexColor; // specify a color output to the fragment shader

void main()
{  gl_Position = vec4(aPos, 1.0); // see how we directly give a vec3 to vec4's constructor
   vertexColor = vec4(0.5, 0.0, 0.0, 1.0); // set the output variable to a dark-red color
}
```

output四維顏色

這個一定要放：收進的position，output會變甚麼

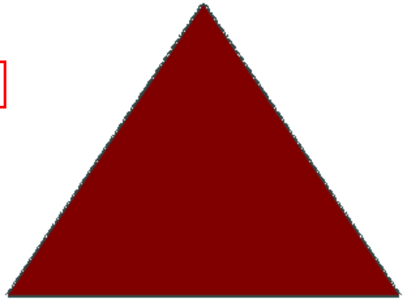
收集到的顏色直接往下塞

Fragment shader code

```
#version 330 core
out vec4 FragColor;

in vec4 vertexColor; // the input variable from the vertex shader

void main()
{   FragColor = vertexColor;
}
```

一定要宣告

in/out要一致



Note: **gl_FragColor** is deprecated   The example is modified from samples in learnopengl.com

Another simple example 基本上照抄

Codes in main.cpp

```
// For the vertex shader
  int vertexShader = glCreateShader(GL_VERTEX_SHADER);
  glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
  glCompileShader(vertexShader);
..........
// For the fragment shader
  int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
  glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
  glCompileShader(fragmentShader);
............
// link the above shaders
  int shaderProgram = glCreateProgram();
  glAttachShader(shaderProgram, vertexShader);
  glAttachShader(shaderProgram, fragmentShader);
  glLinkProgram(shaderProgram);
......
```

從某個character string而來

=>在執行前compile

一般從file讀進來(或character string而來)

連接到主程式

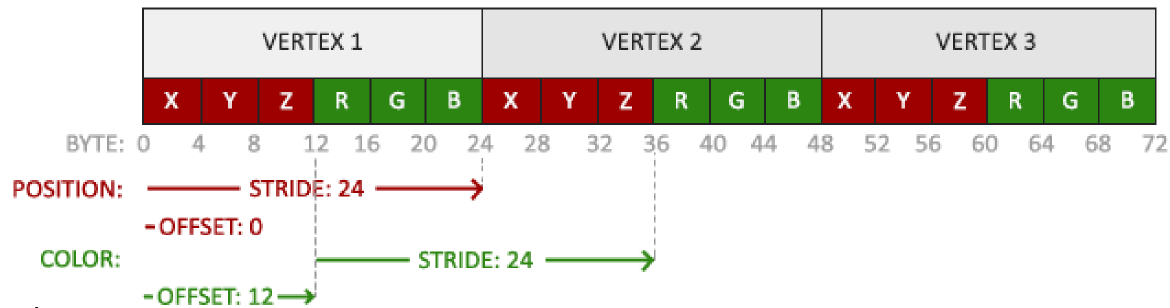
The example is modified from samples in learnopengl.com

Another simple example (cont.)

Codes in main.cpp

VAO : vertex array buffer
VBO : 實際在放的buffer data(EX : vertex shader)

```
float vertices[] = {
    // positions          // colors
     0.5f, -0.5f, 0.0f,   1.0f, 0.0f, 0.0f,   // bottom right
    -0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f,   // bottom left
     0.0f,  0.5f, 0.0f,   0.0f, 0.0f, 1.0f    // top
};
unsigned int VBO, VAO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glBindVertexArray(VAO);
```




```
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

// position attribute
```
position/element/float//每次往後跳，要經過6float(offset要自己算)/起始點從0開始

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
```
*放array和offset要小心

```
// color attribute
```
position/element/float//每次往後跳，要經過6float(offset要自己算)/起始點從第三格開始

```
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);
..........................
glUseProgram(shaderProgram);
```
=>選擇要使用哪個shader

Another simple example (cont.)

Vertex shader code

```
#version 330 core
layout (location = 0) in vec3 aPos;   // the position variable at position 0
layout (location = 1) in vec3 aColor; // the color variable at position 1
```

0.5, -0.5, 0

0, 1, 0

每個點會獨立收到一個不同的資料 attributePosition/attributeColor

```
out vec3 ourColor; // output a color to the fragment shader

void main()
{  gl_Position = vec4(aPos, 1.0);
   ourColor = aColor; // set ourColor to the input vertex color
}
```

out attribute color會做內插
(vs. phong shading : 用out normal做內插)

Vert buffer passed from main.cpp

```
float vertices[] = {
     // positions        // colors
     0.5f, -0.5f, 0.0f,  1.0f, 0.0f, 0.0f,
    -0.5f, -0.5f, 0.0f,  0.0f, 1.0f, 0.0f,
     0.0f,  0.5f, 0.0f,  0.0f, 0.0f, 1.0f
};
```

Fragment shader code

每個fragment都會平行執行fragment shader，
自己只知道自己的資料(點)，只能動顏色、不能動位置

```
#version 330 core
out vec4 FragColor;
in vec3 ourColor;
```

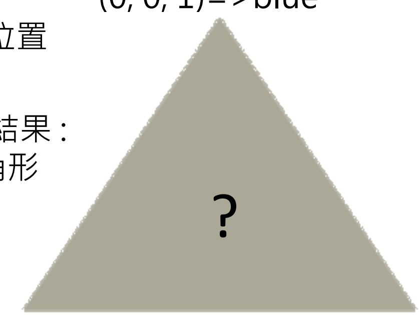
內插完的結果

```
void main()
{   FragColor = vec4(ourColor, 1.0);
}
```

直接塞color轉成四維

Rasterize結果：
漸層式三角形
(三色漸變)
(三點RGB)

(0, 0, 1)=>blue



(0, 1, 0)=>green (1, 0, 0)=>red

An example with matrix passing

Codes in main.cpp

Data passing through
***uniform** variables* *glmathlib

```
float vertices[] = {
      // positions          // colors
       0.5f, -0.5f, -1.5f,  0.5f, 0.0f, 0.0f,  // Mid right
      -0.5f, -0.5f, -1.5f,  0.5f, 0.0f, 0.0f,  // Mid left
       0.0f,  0.5f, -1.5f,  0.5f, 0.0f, 0.0f,   // top
       -0.5f, -0.5f, -1.5f,  0.0f, 0.5f, 0.0f,  // Mid left
       0.5f, -0.5f, -1.5f,  0.0f, 0.5f, 0.0f,  // Mid right
       0.0f,  -1.5f, -1.5f,  0.0f, 0.5f, 0.0f   // bottom
};
..........................
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glm::mat4 projection_matrix = glm::perspective(glm::radians(60.0f), (float)SCR_WIDTH / (float)SCR_HEIGHT,
0.1f, 10.0f);
glUniformMatrix4fv(glGetUniformLocation(shaderProgram, "projection"), 1, GL_FALSE,
glm::value_ptr(projection_matrix) );
```

link到shader code的變數名稱

另一種寫法：&projection_matrix(0)(0);

```
glm::mat4 model_matrix = glm::mat4(1.0f); // make sure to initialize matrix to identity matrix first
model_matrix = glm::rotate(model_matrix, glm::radians(15.0f), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(glGetUniformLocation(shaderProgram, "model"), 1, GL_FALSE, &model_matrix[0][0] );
```

原來的matrix再乘matrix(model沿z軸逆時針旋轉15度)

送到shader

```
glDrawArrays(GL_TRIANGLES, 0, 6);
```

6 : 6個頂點=>兩個三角形(頂點重複)

EBO : 只要列4個頂點

An example with matrix passing (cont.)

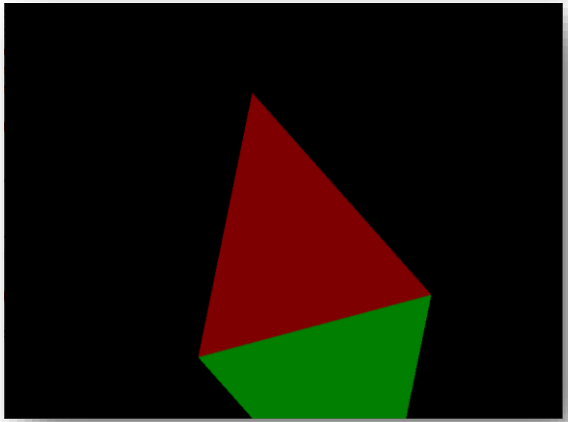
Vertex shader code

先傳誰進來沒差，乘的順序對就好

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aColor;
out vec3 ourColor;
uniform mat4 model;
uniform mat4 projection;

void main()
{  gl_Position = projection * model * vec4(aPos, 1.0);
   ourColor = aColor;
 }
```

Fragment shader code

```
#version 330 core
out vec4 FragColor;
in vec3 ourColor;
void main()
{
  FragColor = vec4(ourColor, 1.0f);
}
```



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
    - 堆疊
      Color material application and color clamping
    - Texture coordinate generation

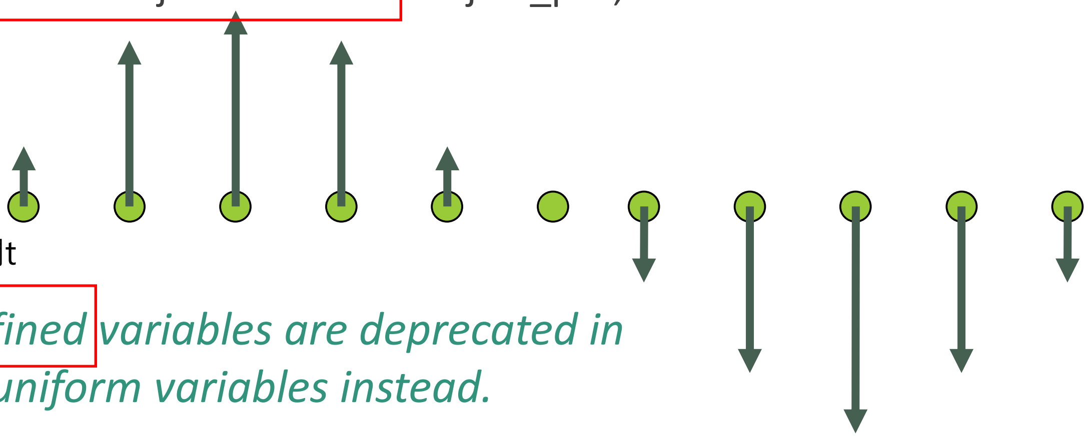
Vertex Shader Applications

- 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()
{vec3 object_pos;
float s;
s = 1.0 + 0.1*sin(xs*time)*sin(zs*time);  用layout傳 : 每個點都要拿到不同的sin的點
object_pos = aPos;
object_pos.y = s* aPos.y;  現在禁用(因為效率造成影響)
gl_Position = gl_ModelViewProjectionMatrix* object_pos;
}
```



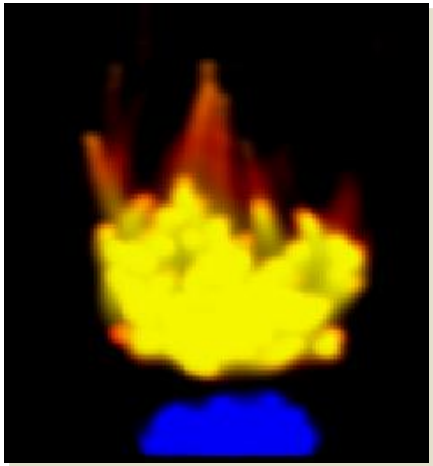
改變時間t

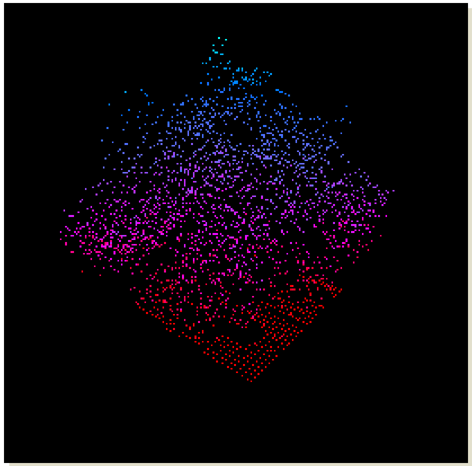
Note: Several gl_ predefined variables are deprecated in the newer version. Use uniform variables instead.

Applications: Particle Systems

Uniform: passing parameters to vertex and fragment shaders.

```
……………………………
uniform vec3  vel;
uniform float g, m, t;
void main()  只變更時間t
{
vec3 object_pos;
object_pos.x = aPos.x + vel.x*t;
object_pos.y = aPos.y + vel.y*t + g/(2.0*m)*t*t;
object_pos.z = aPos.z + vel.z*t;
gl_Position = gl_ModelViewProjectionMatrix*
vec4(object_pos,1);
}
```





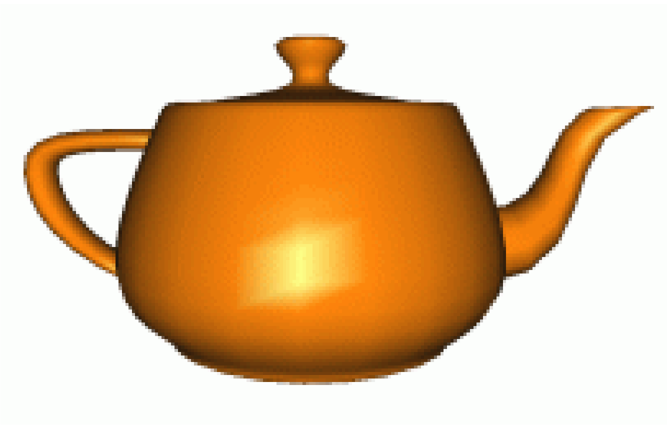
Note: Several gl_ predefined variables are deprecated in the newer version. Use uniform variables instead.

Fragment Shaders

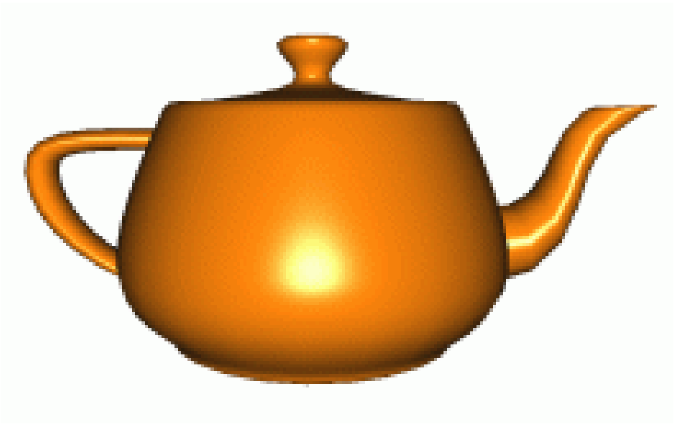
- What is a fragment? 可能很多點疊在一起，但最終只會在螢幕上顯示一個點
  - Cg Tutorial says: “You can think of a fragment as a ‘potential pixel’”
- Perform per-pixel calculations
  - Without knowledge about other fragments (parallelism)
- Your program’s responsibilities:
  - Operations on interpolated values
  - Texture access and application
  - Other functions: fog, color lookup, etc.

Fragment Shader Applications

(Per-pixel) Phong shading 每個三角形裡都要算顏色



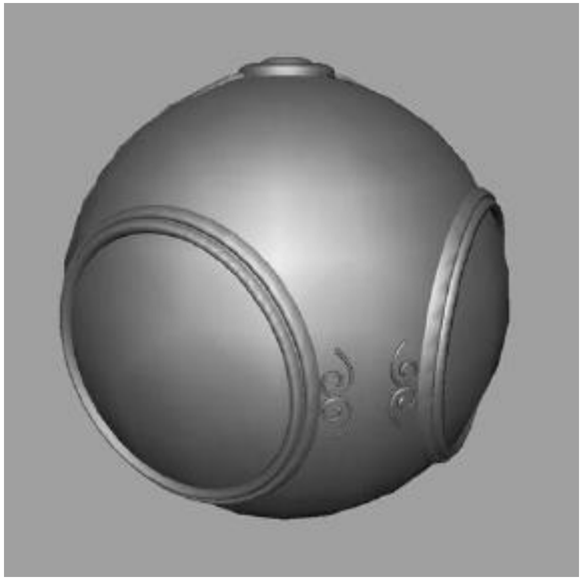
Per-vertex lighting



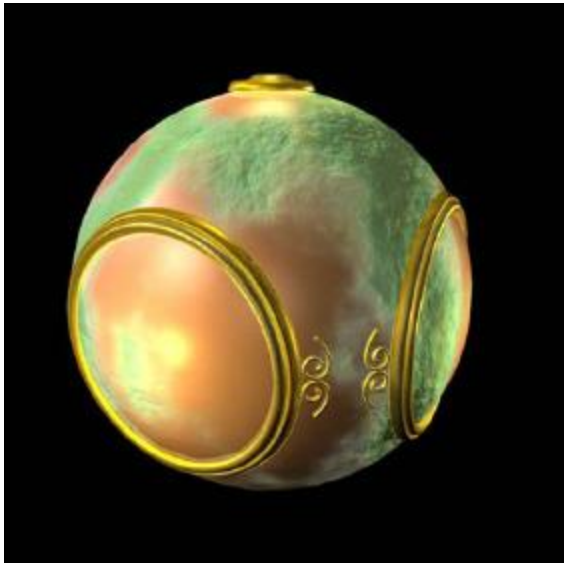
Per-fragment lighting

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

Fragment Shader Applications



smooth shading



bump mapping

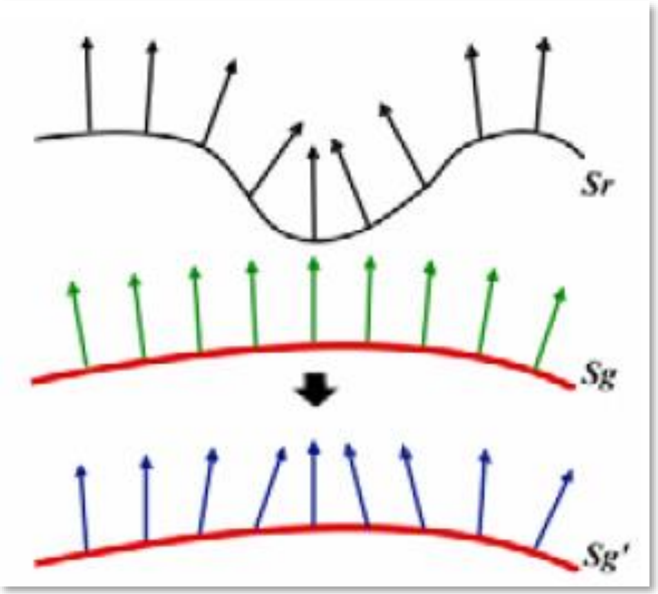
Bump Mapping

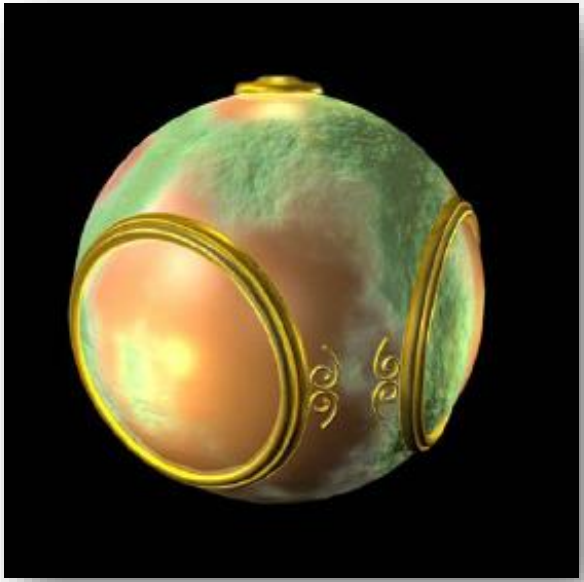
擾動

- Perturb normal for each fragment

- Store perturbation as textures

normal





節省texture memory、顏色階段化

Toon Shading

卡通渲染

現在被禁用惹

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(); }
```

先做phong shading壓成三種顏色

要往下傳norma

=>改用projection * view * model * apos

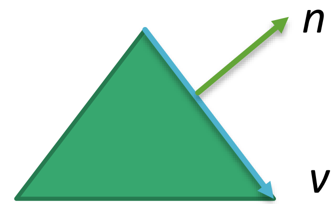
- The fragment shader becomes

```
………………………
in vec3 vnormal;
out vec4 FragColor;
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);
    FragColor = color; }
```

也被禁用=>改light * normal

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

gl_NormalMatrix



$n \cdot v = 0$

*自己想旋轉 : normal * rotation

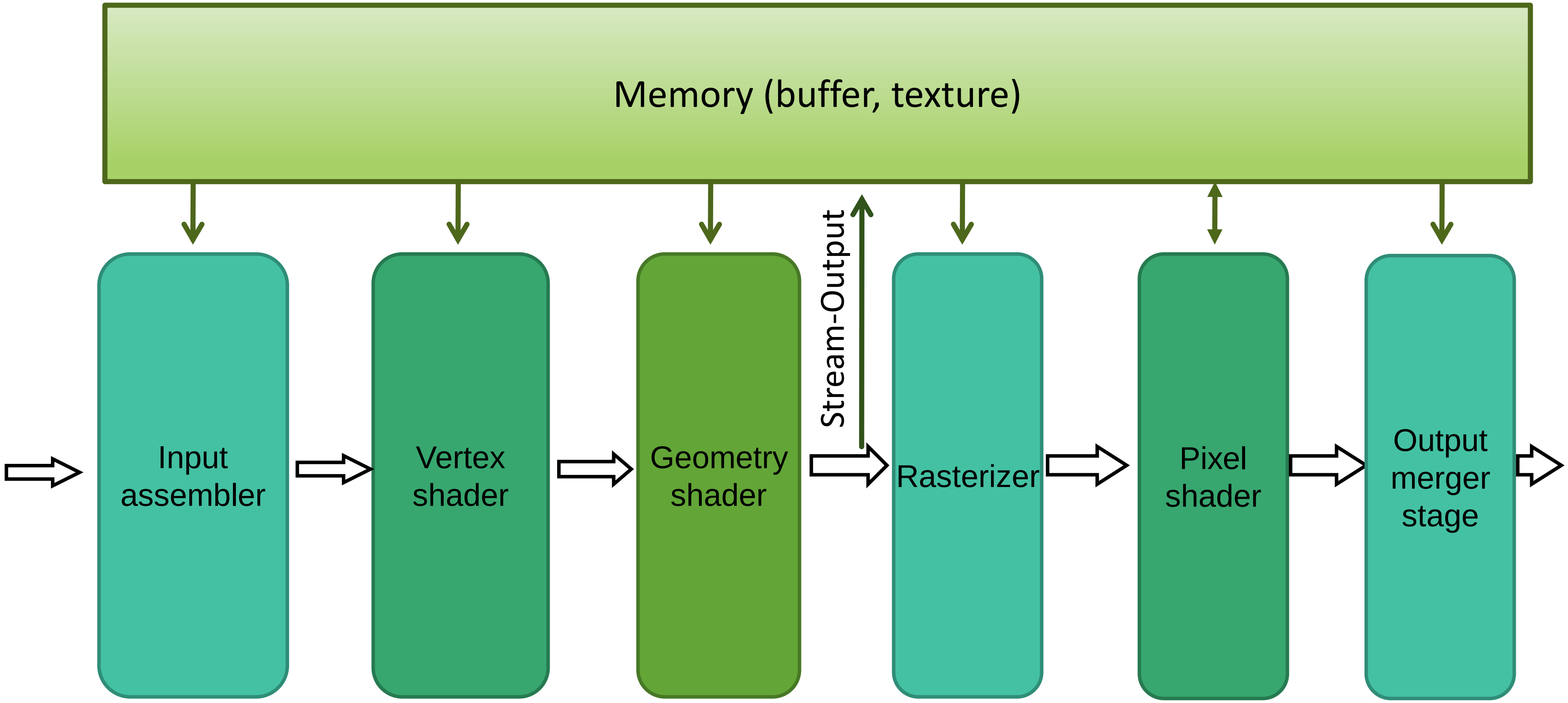
- Can we directly apply the modelview matrix M to a normal vector ?
  - Problem: If the upper-left 3x3 submatrix M_s is not orthogonal, $n' = M_s n$ is not perpendicular to $v' = M_s v$

*Our goal is to find a matrix N for n'' dot v' = 0, where n'' = Nn. Nn未知
*(Nn) dot (Msv) = 0 = (n^t)(N^t)Msv
*It is reasonable to choose (N^t)Ms = I, since (n^t)v = 0

互為inverse

N = (Ms)^(-1)^t

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.

    放大
  - The Geometry shader supports limited geometry amplification and de-amplification. (discard the primitive, or emit one or more new primitives)

    一次可以看整個三角形或邊，也可以多加點
  - E.g. Subdivision, point ->billboard, silhouette edge -> fur, etc.

    subdivision：一個三角形拆成三個三角形 邊緣長毛
- **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.)

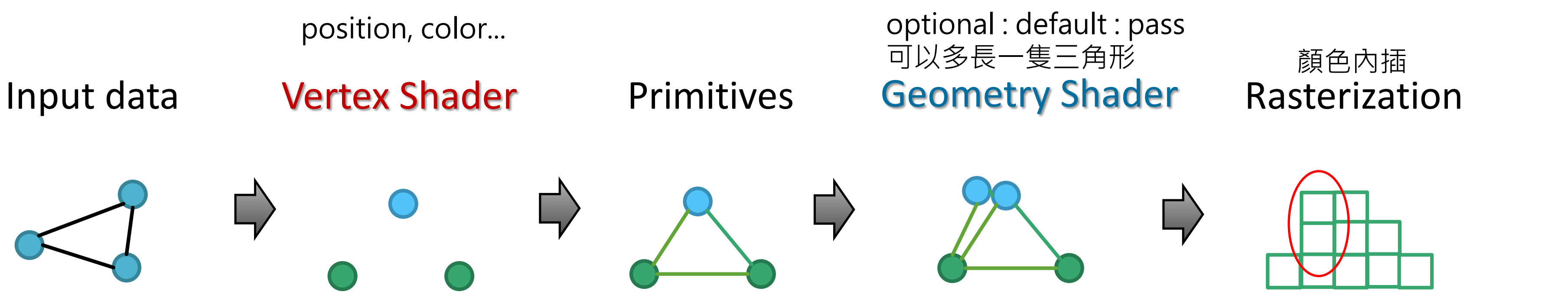
position中間做內插

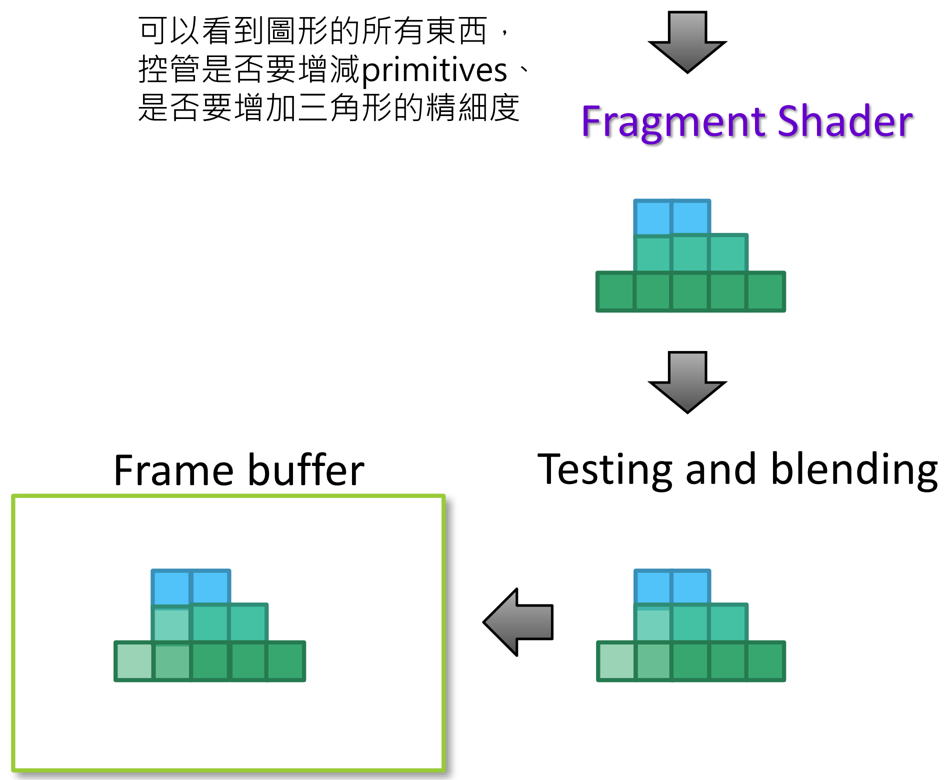
- **Rasterizer**: clips primitives, prepares primitives for the pixel shader and determines how to invoke pixel shaders.

- **Pixel shader**: receives interpolated data for a primitive and generates per-pixel data, such as color.

- **Output-merger stage**: 三角形做混合或把不用的東西砍掉
  - combines various types of output data (pixel shader values, depth and stencil 模板 information) with the contents of the render target and depth/stencil buffers to generate the final pipeline result.

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





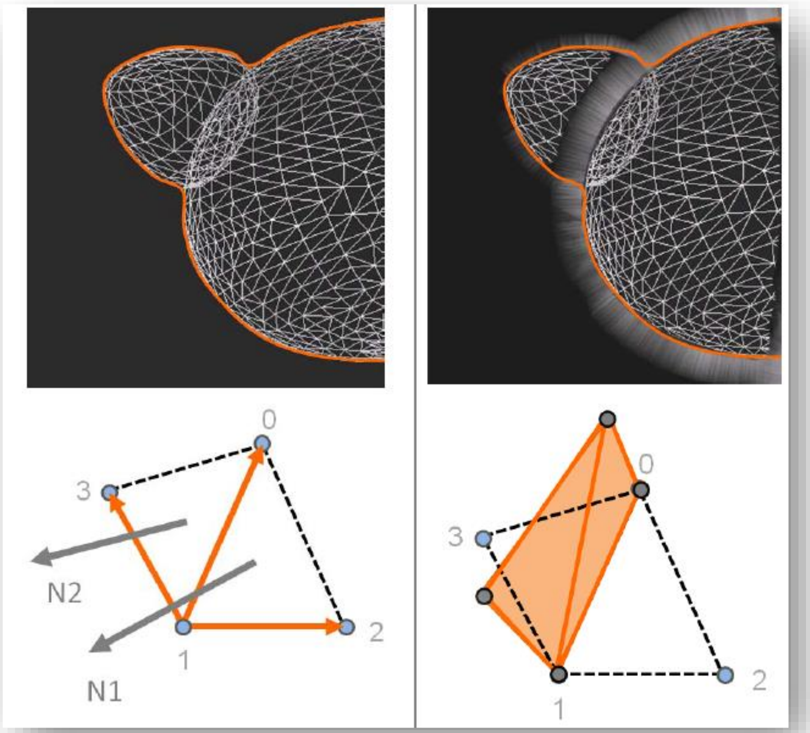
Vertex shaders
per-vertex functions

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

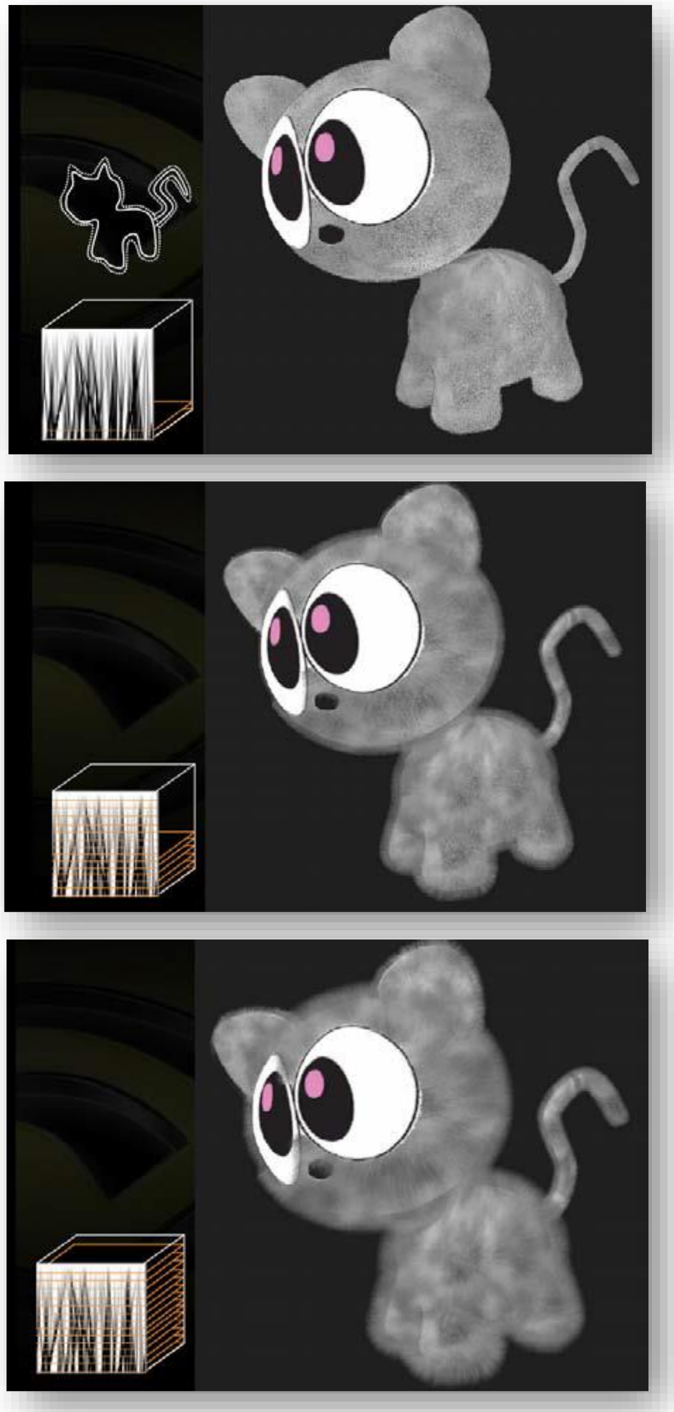
Fragment shaders
per-fragment (pixel) function.

D3D 10 Pipeline (cont.)

細毛



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



vertex=>geometry=>fragment

Previous example using Geometry Shader

Vertex shader code

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aColor;
out VS_OUT{
   vec3 vsColor;
} vs_out;
uniform mat4 model;
uniform mat4 projection;

void main()
{ gl_Position = projection * model * vec4(aPos, 1.0);
  vs_out.vsColor = aColor;
 }
```

兩個三角形一邊(兩點)重疊

Geometry shader code

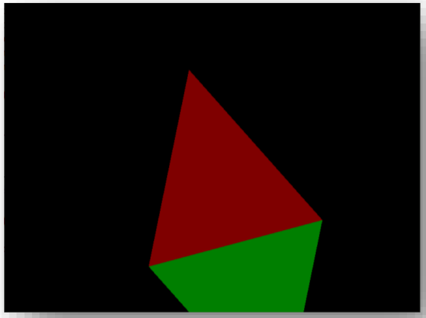
```
#version 330 core
layout (triangles) in;
layout (triangle_strip, max_vertices = 3) out;
in VS_OUT{
    vec3 vsColor;
} gs_in[];
out vec3 ourColor;

void main()
{
  for(int i=0; i<3; i++)
  { gl_Position = gl_in[i].gl_Position;
    ourColor = gs_in[i].vsColor;
    EmitVertex();
  }
  EndPrimitive();
}
```

得到兩個分開的三角形

把資料收進來，normal可以在這裡算

Fragment shader code

```
#version 330 core
out vec4 FragColor;
in vec3 ourColor;
void main()
{   FragColor = vec4(ourColor, 1.0f);
}
```



input 3 vertex, 1 triangle=>input 4 vertex, 2 triangle

Adding additional triangles with GS

Geometry shader code

可以省bandwidth

```
#version 330 core
layout (triangles) in;                          新增一個點
layout (triangle_strip, max_vertices = 4) out;
in VS_OUT{
    vec3 vsColor;
} gs_in[];
out vec3 ourColor;

 void main()  把資料抓進來(原始三個點)
{  for(int i=0; i<3; i++)
  {gl_Position = gl_in[i].gl_Position;
   ourColor = gs_in[i].vsColor;
   EmitVertex(); 把資料吐出去
  } normalized space        vertex
  gl_Position = gl_in[0].gl_Position + vec4(0.2f, 0.2f, -0.2f, 0.0f);
  ourColor = vec3(0.0f, 0.0f, 0.8f);                在這個地方多塞一個點
  EmitVertex();
  EndPrimitive();
}
```

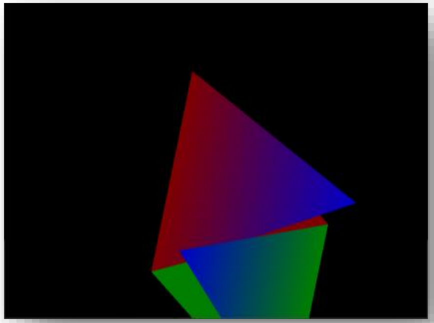
Note:
Triangle strip: v0, v1, v2, v3
$\Rightarrow$ Triangle 1 (v0, v1, v2)
$\Rightarrow$ Triangle 2 (v1, v2, v3)

projection * model matrix => static constant
在原始空間操作完(EX : 加一個點)，
再放Geometry shader

uniform可以送給所有shader

For each triangle, add one additional triangle.

多長了一個斜斜的三角形



D3D 11 Pipeline

- In D3D10, the Geometry shader may subdivide the surfaces by multiple passes.

把東西細化，把三角形割碎

- 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.

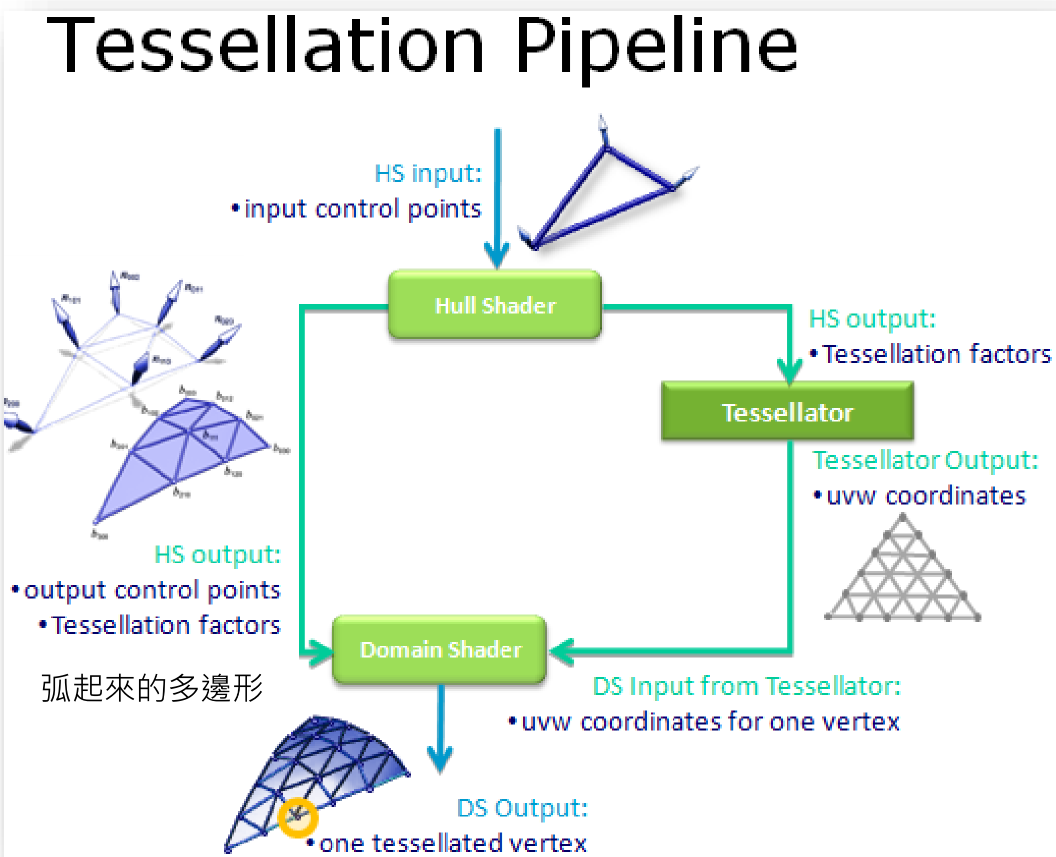


Figure from: developer.download.nvidia.com/presentations/2009/GDC/GDC09_D3D11Tessellation.pdf

input assmbler

vertex shader

tessellation

geometry shader

rasterization

fragment shader

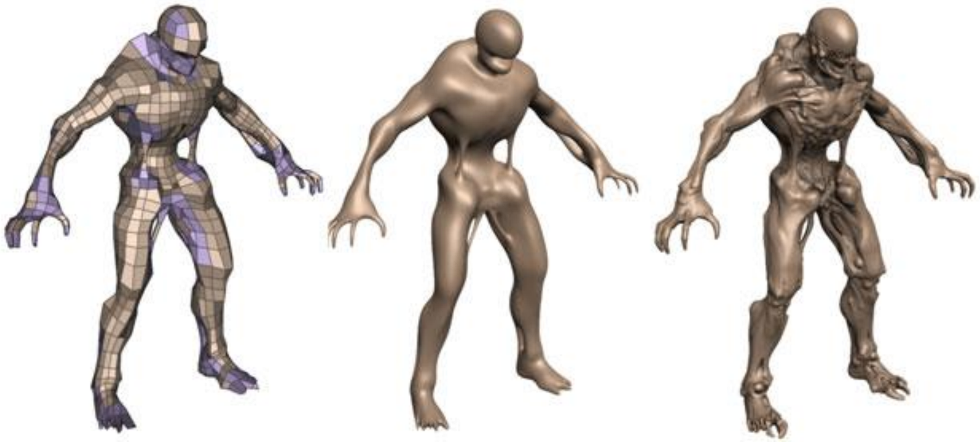
color blending

framebuffer

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

D3D 11 Tesselation

Model refinement



Tessellation with displacement mapping

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

End of Chapter 6