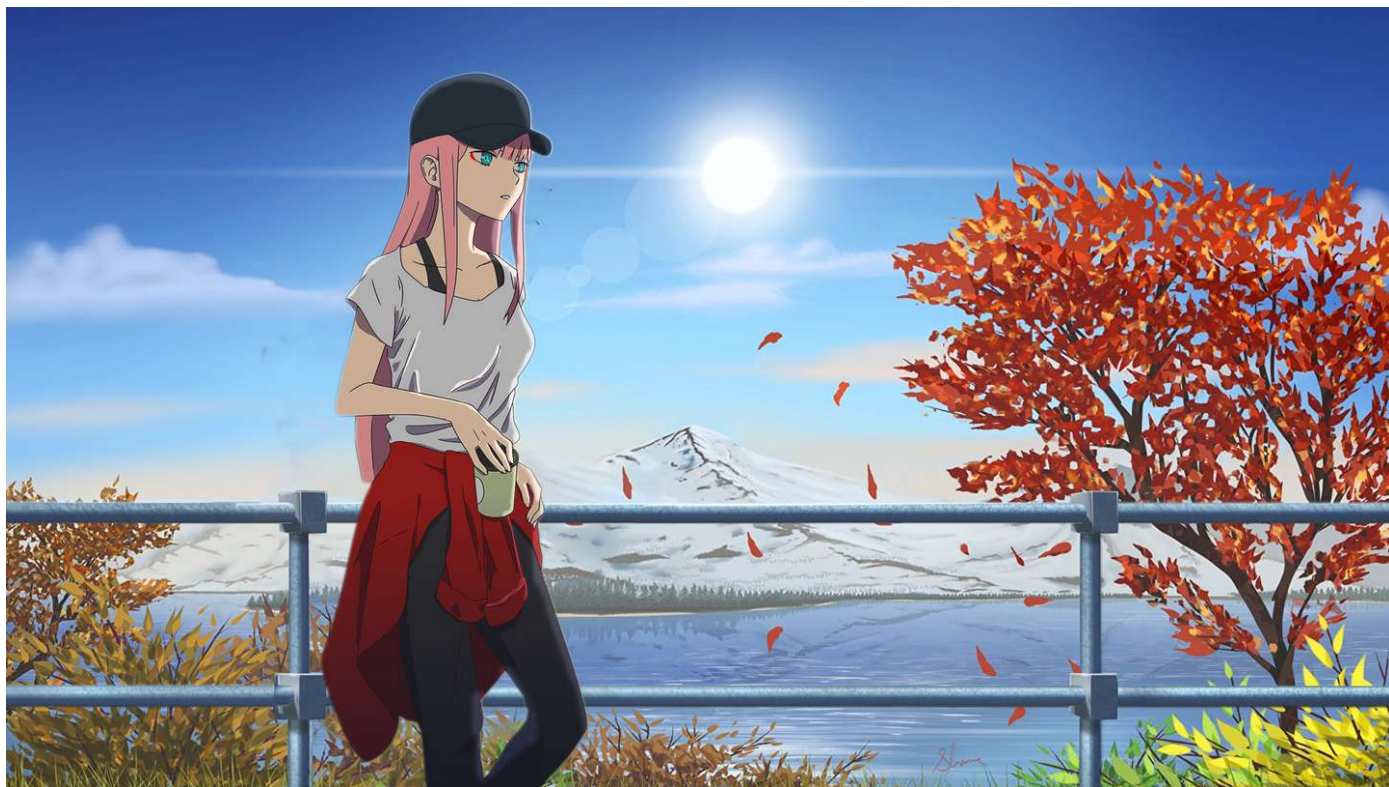


# GAMES202 高质量实时渲染课程笔记Lecture 2- Recap of CG Basics



本文是games202的Lecture 2的笔记,如有任何错误之处请各位大佬指正.

[GAMES202-高质量实时渲染\\_哔哩哔哩\(°-°\)つロ 干杯~-bilibili](#)

本课目录

- Recap of CG Basics
- Basic GPU hardware pipeline (复习GPU渲染管线)
- OpenGL and OpenGL Shading Language (opengl和对应的着色语言--GLSL)

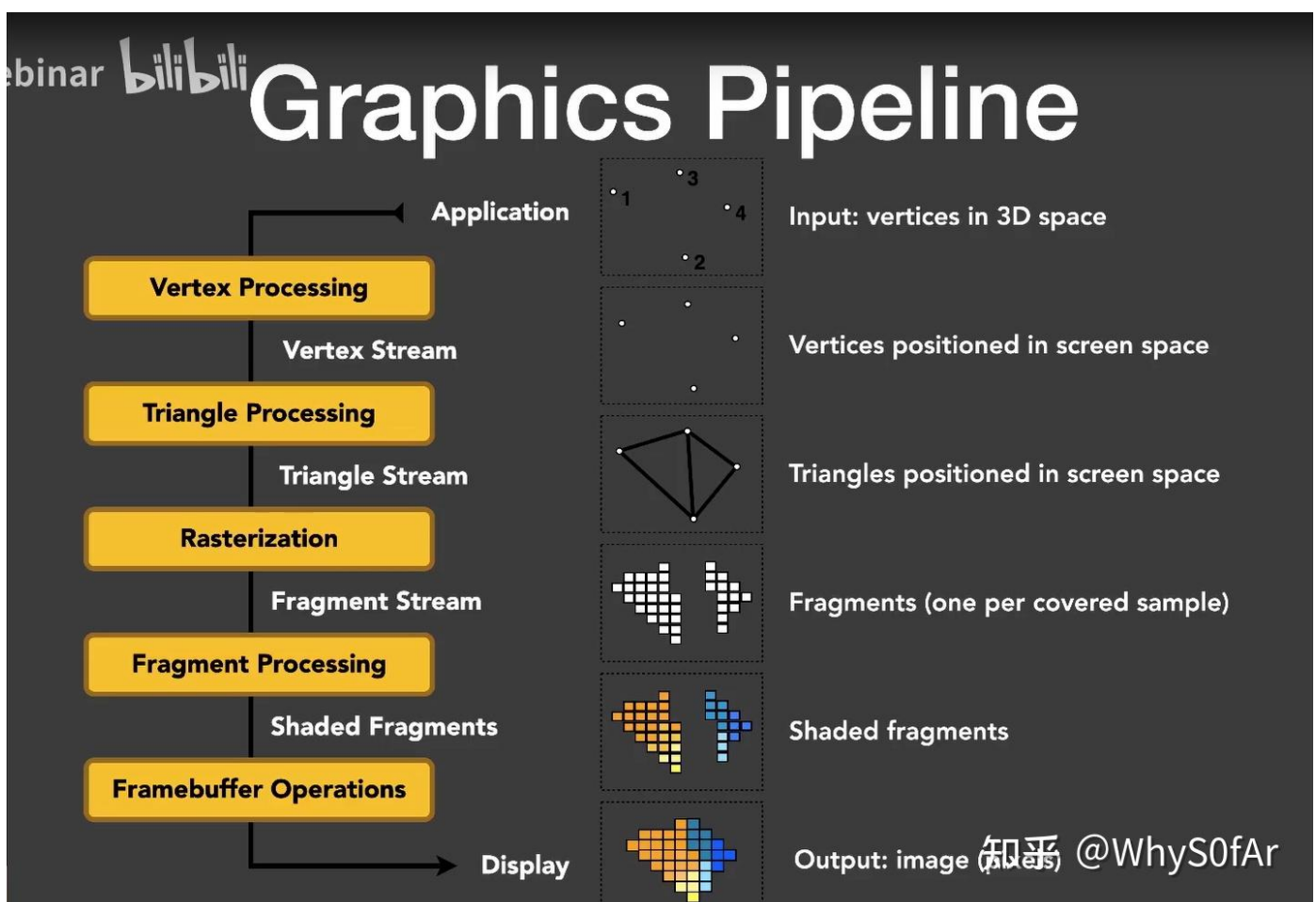
- The Rendering Equation
- 相关微积分知识

## I -> Graphics (Hardware) Pipeline

首先我们对渲染管线进行一个复习:

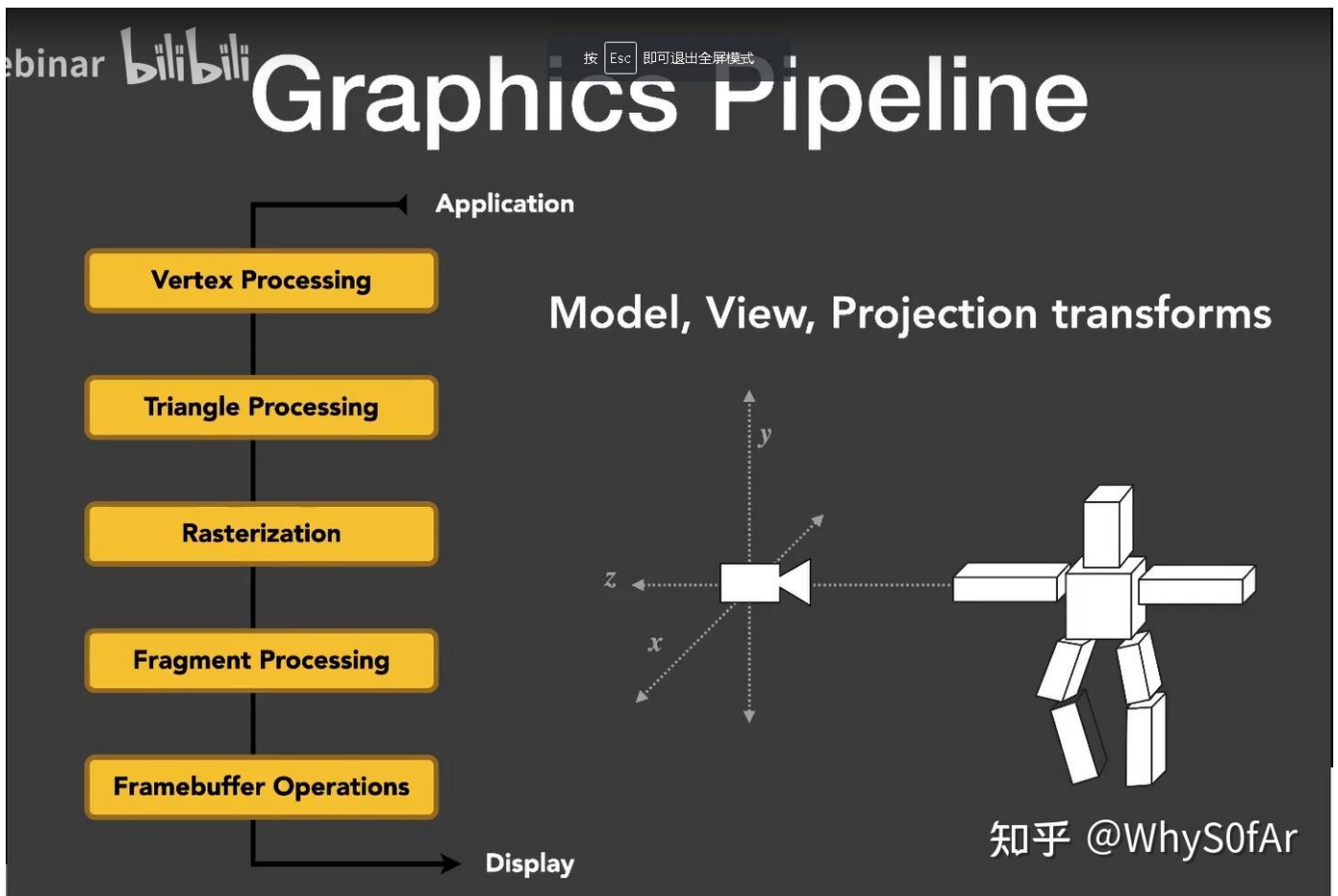
我们知道物体在最开始是3D的模型,我们经过渲染后最终会得到一张图,而这中间经过了一些系列的操作,这个过程我们称其为渲染管线(渲染流水线).

渲染管线的一系列操作是在现代GPU上完成的,因为GPU完成或者说是计算的快,因为GPU的并行度高.



## 渲染管线图

3D物体在空间中以一堆点和点的连接关系 来表示,每个点都会经过第一个步骤**顶点处理**,也就是经过一系列的变换,如**MVP变换**,**ViewPoint变换**,从而变换成屏幕上的一个点.

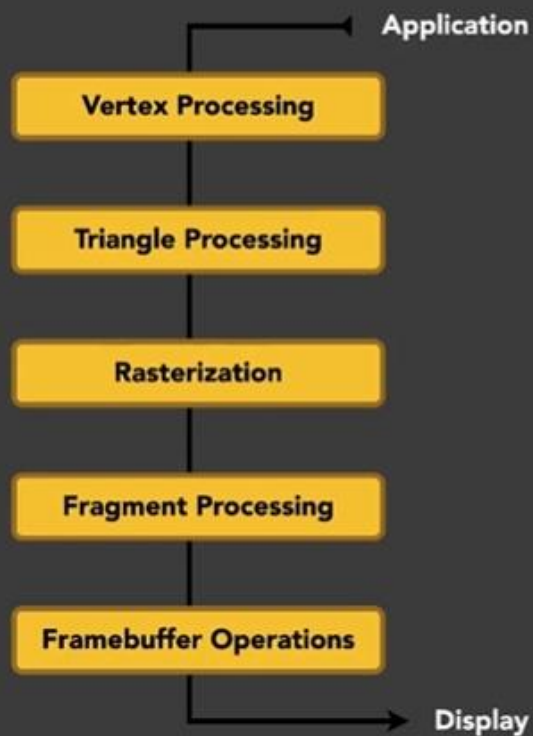


## 顶点处理

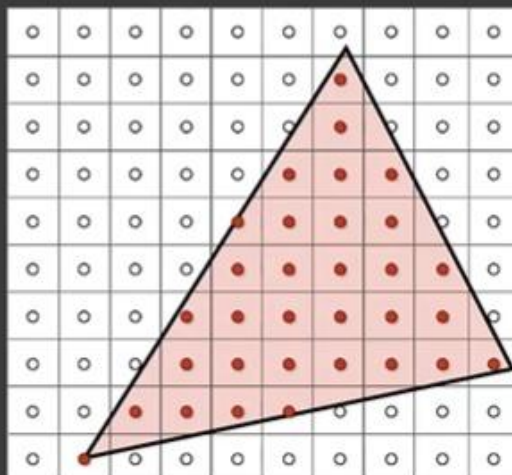
经过这一些系列的变换之后,各点之间的连接关系是不会变换的,只是点被投影到了屏幕上,因此变换前的三个点所组成的三角形在变换后仍由这三个点组成三角形.

三角形投影到屏幕上后我们要进行光栅化操作来将三角形离散成一堆**像素(或者是fragments)**.

# Graphics Pipeline



## Sampling triangle coverage

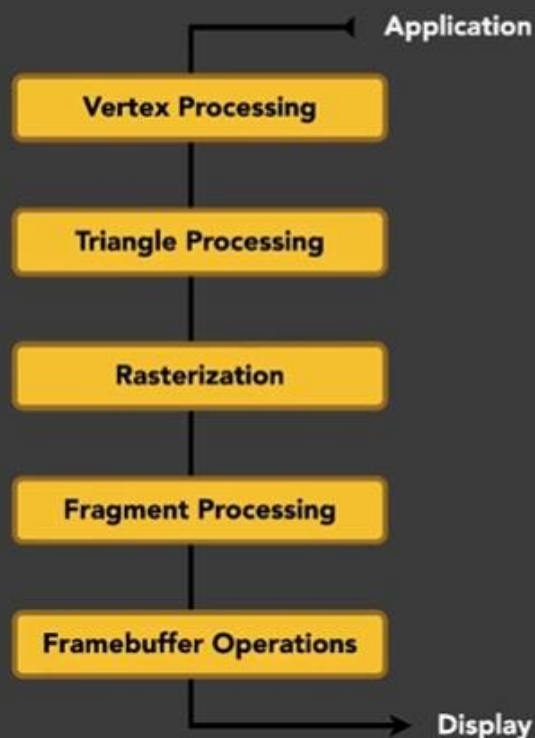


知乎 @WhyS0fAr

## 光栅化

在打散成像素的过程中我们要利用深度缓存来处理遮挡问题.

# Graphics Pipeline



## Z-Buffer Visibility Tests



知乎 @WhyS0fAr

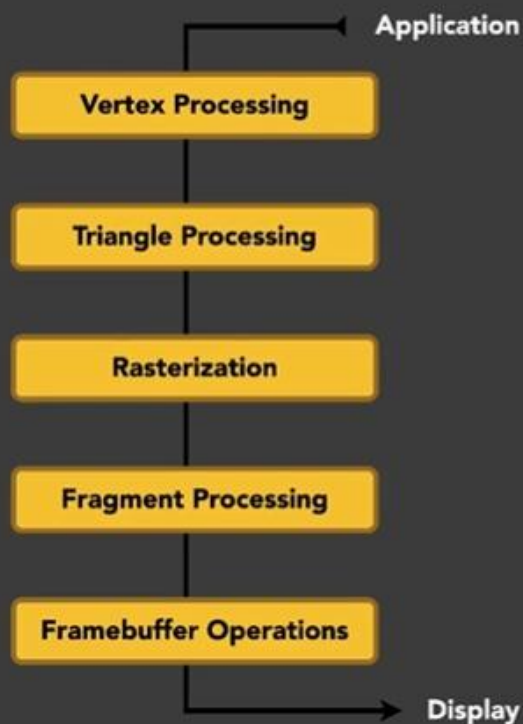
## 深度缓存

接着我们进行着色处理,也就是计算它应该长什么样子,比如布林肯着色模型.

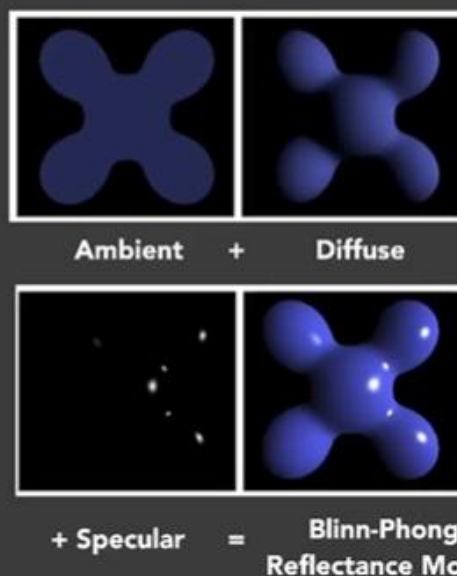
布林肯是一个经验式的模型,因此它并不是完美的,它对全局或者间接光照是一个近似的做法,它对全局的一些现象处理的不好,比如阴影,光线的多次弹射.



# Graphics Pipeline



## Shading



知乎 @WhyS0fAr

在着色的过程中,我们可以在三角形内部或者是整个物体上的任意一个地方得到一个纹理的坐标,从而把一张图给贴在上面,或者根据它的纹理坐标,任一个位置我们都可以知道在纹理的哪个地方去查询.

同时我们还提到了一个叫**插值**的概念,就是说我知道三角形三个顶点对应的纹理坐标,我们可以通过插值来在三角形内部的任一位置得到一个平滑的纹理过渡.

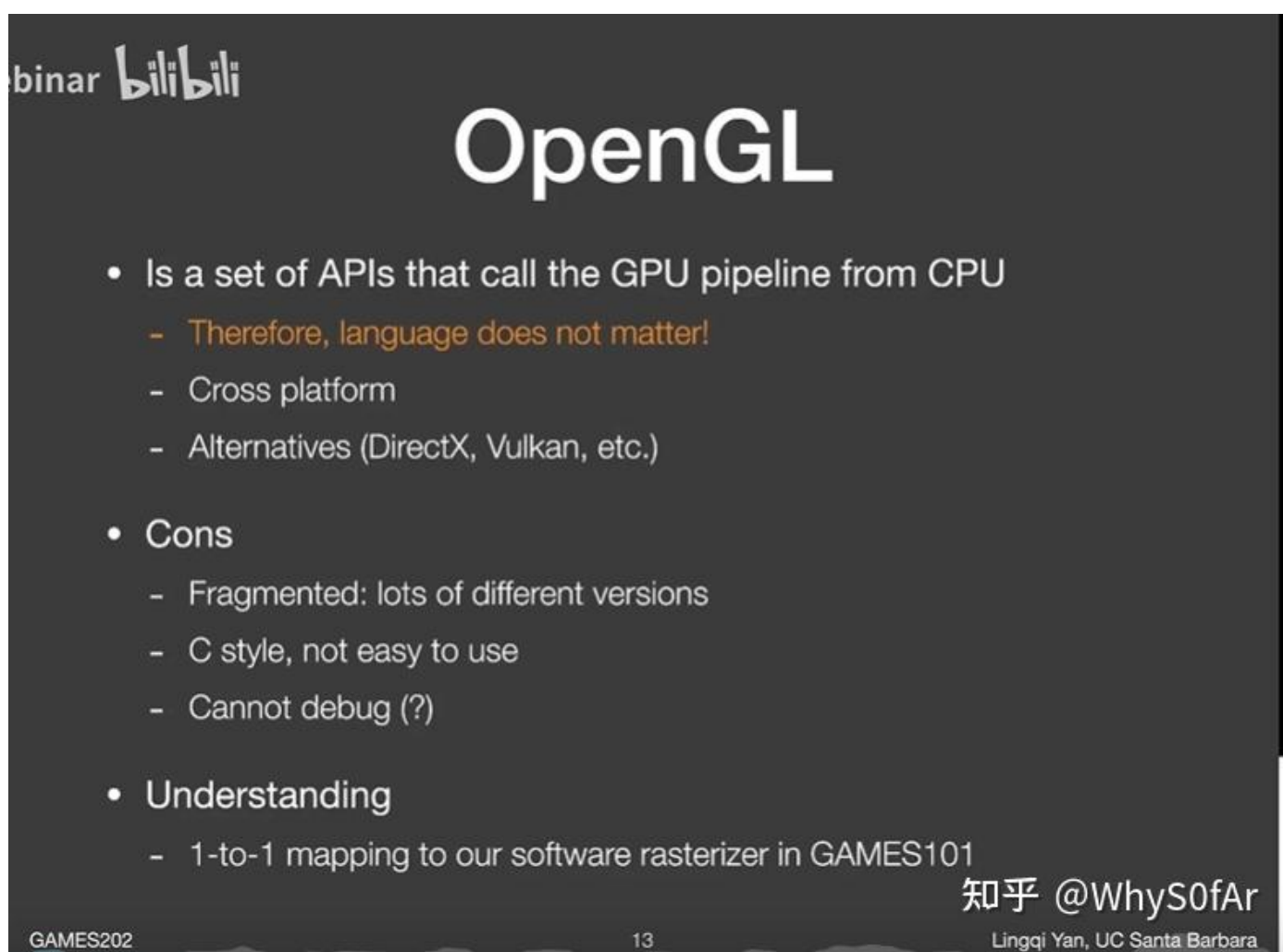
最终我们得到一张成图.

## II -> OpenGL

首先什么是Opengl:

opengl是一系列在cpu端上负责调动gpu的api, 也就是在CPU端写的,我们可以知道opengl用什么语言写是没有关系。

我们更关心的是GPU怎么去执行,而不是CPU如何让GPU去执行.



binar bilibili

# OpenGL

- Is a set of APIs that call the GPU pipeline from CPU
  - Therefore, language does not matter!
  - Cross platform
  - Alternatives (DirectX, Vulkan, etc.)
- Cons
  - Fragmented: lots of different versions
  - C style, not easy to use
  - Cannot debug (?)
- Understanding
  - 1-to-1 mapping to our software rasterizer in GAMES101

知乎 @WhySofAr

GAMES202 13 Lingqi Yan, UC Santa Barbara

OpenGL的一个特点:

可以跨平台。

## OpenGL的缺点:

1. 版本过于碎片化
2. c风格语言，不是很好用
3. 几年前不好debug，现在方便很多

## OpenGL就像画油画:

我们要渲染一个3D的场景,我们类比画油画的各个步骤:

1. 摆好物体和模型

binar bilibili

# OpenGL

- A. Place objects/models
  - Model specification
  - Model transformation
- User specifies an object's vertices, normals, texture coords and send them to GPU as a Vertex buffer object (VBO)
  - Very similar to .obj files
- Use OpenGL functions to obtain matrices
  - e.g., glTranslate, glMultMatrix, etc.

知乎 @WhySofAr  
Lingqi Yan, UC Santa Barbara

GAMES202 15

在摆好物体和模型之前我们会有两个问题:



### **a)我们需要告诉GPU我们要用什么样的模型**

在OPENGL种我们通过VBO我们可以把需要渲染的图元的顶点信息，直接上传存储在GPU的显存中.

VBO就是GPU种一块用来存储你的模型的区域,模型的存储方式与存储三角形的OBJ方式相似,存储一堆 vertices,normals,texture coords等以一定的格式组织了一个物体应该放在GPU的何处.

### **b)模型该如何摆放**

在101中我们需要自己手写各种的矩阵来进行物体的运动,而在OPENGL中内置了这些函数,我们只需要调动函数写入参数即可,不用再自己去写函数.

## **2.架好画架**

# OpenGL

- B. Set up an easel
  - View transformation
  - Create / use a framebuffer
- Set camera (the viewing transformation matrix) by simply calling, e.g., gluPerspective

```
void gluPerspective( GLdouble fovy,  
                    GLdouble aspect,  
                    GLdouble zNear,  
                    GLdouble zFar);
```

架好画架也会遇到两件事:

## a) view transformation(视图变换)

首先放置相机，视图变换就是相机选用的是透视投影还是正交投影,在101中仍是我们自己去推导和写的矩阵,而opengl的api简化了摄像机的视图变化,让一切变得简单了许多.

我们只需要规定相机的一些属性就可以了,比如说:

fov(可视角度),aspect(长宽比),zNear(近平面),zFar(远平面).

## b) creat / use a framebuffer(建立画架)


要建立opengl的画架，就是framebuffer,为了要使用一个画架,我们一定要在上面贴一块画布的,因此到第三步.

### 3.在画架上贴上一块画布

binar bilibili

OpenGL

- C. Attach a canvas to the easel
- Analogy of oil painting:
  - E. you can also paint multiple pictures using the same easel
- One rendering **pass** in OpenGL
  - A framebuffer is specified to use
  - Specify one or more textures as output (shading, depth, etc.)
  - Render (fragment shader specifies the content on each texture)



GAMES202

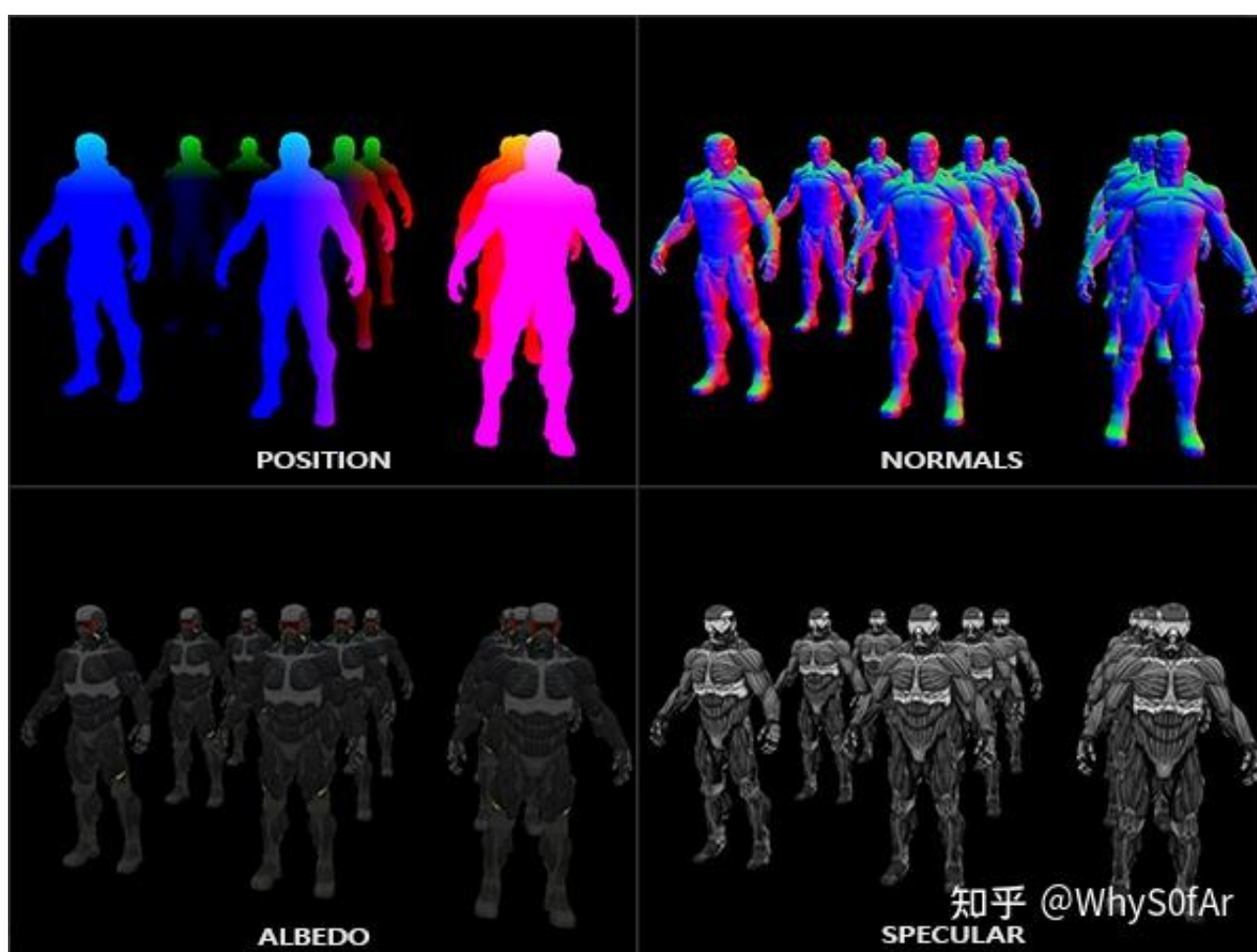
17

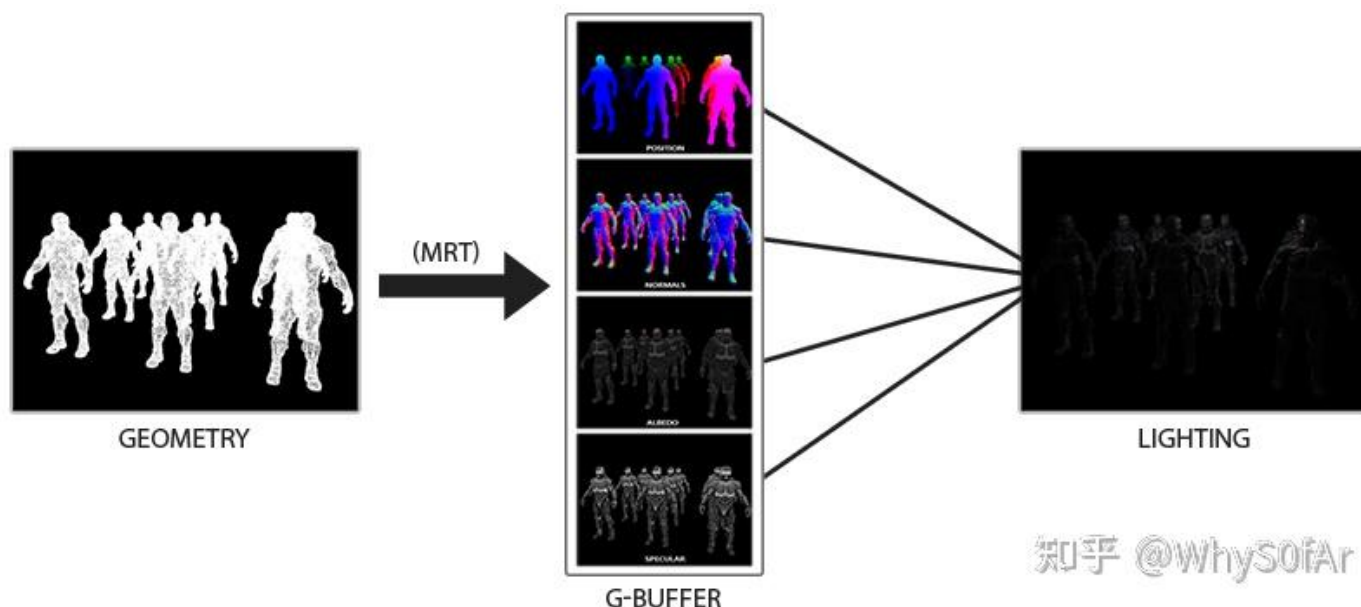
知乎 @WhySofAr  
Lingqi Yan, UC Santa Barbara

framebuffer对场景渲染一次，可以用渲染出好几张不同性质的纹理，（也解释了第5点画好一张后换张画布可以继续画），最后由fragment shader 告诉你要写到哪一个纹理上去。

也就是你从一个角度看过去渲染一次场景可以输出好几张不同的图,可能有一张是shading的结果,有一张是深度.

例如这几张图就是通过MRT在对场景渲染一次后得到的四张不同性质的图:





我们也是可以直接把framebuffer渲染的结果显示到屏幕上的,但是直接把framebuffer的渲染结果放在屏幕上会造成屏幕撕裂,因为你这一帧还渲染一半,下一帧就开始渲染覆盖你这一帧所看到的内容,从而导致了屏幕撕裂.

在游戏中时常有一个叫做垂直同步的设置能解决这个问题。或者先把渲染好的结果存储在一个纹理或者缓冲区里,确定没问题在放在屏幕上,这种方法叫做双重缓冲,更复杂的还有三重缓冲。

#### 4.在画布上绘画

我们要把场景给画在画布上,那么无论如何我们肯定要要进行 shading,那么:



如何做shading呢?(我们在101中已经讲的很清楚了)

课程中只会用到顶点着色和像素着色。

顶点着色的话,每个顶点的要在顶点着色器操作, 首先要进行mvp的投影,之后我们将需要做插值的值进行插值得到结果后,再将结果传递到像素着色器。

那么像素着色器得到的输入就是顶点着色器的输出在一个顶点上的属性,接着就会被插值到任何一个像素上的属性.

opengl会将三角形打成一堆像素, 然后对每个像素着色。

这门课程最重要的就是我们如何去写顶点着色器和片段着色器.

我们来做一个简单的总结:

# OpenGL

- Summary: in each pass
  - Specify objects, camera, MVP, etc.
  - Specify framebuffer and input/output textures
  - Specify vertex / fragment shaders
  - (When you have everything specified on the GPU) Render!

知乎 @WhyS0fAr

Opengl就是告诉GPU在每一趟渲染中需要:

- 指定物体，相机，mvp矩阵等
- 指定Framebuffer和输/出纹理
- 指定顶点、fragment着色器
- （当在gpu里确定了一切）渲染

## 6.在画新图时可以采用或参考旧画上的内容

之前渲染的纹理可以给之后渲染参考（多pass渲染）.

那么为什么场景要渲染很多次？

我们以shadow map为例,他是一个典型的两趟做法,从light和camera分别去看场景,因此在light得到的深度这么一个纹理,我们可以在camera时候再去用它.

### III -> OpenGL Shading Language (GLSL)

顾名思义就,描述着色器怎么操作的语言就是着色语言,着色语言风格偏c语言。

#### 怎么使用shader

首先要写shader,然后让opengl去编译shader,但是你可能会比编译fragment shader或者是vertice shader,因此可以建立program集合了你写的所有自定义shader,再做一个链接,最后使用链接好的program渲染。

写shader可以近似于在cpu上编程,二者十分相似。

# Shader Initialization Code (FYI)

```
GLuint initshaders (GLenum type, const char *filename) {
    // Using GLSL shaders, OpenGL book, page 679
    GLuint shader = glCreateShader(type) ;
    GLint compiled ;
    string str = textFileRead (filename) ;
    GLchar * cstr = new GLchar[str.size()+1] ;
    const GLchar * cstr2 = cstr ; // Weirdness to get a const char
    strcpy(cstr, str.c_str()) ;
    glShaderSource (shader, 1, &cstr2, NULL) ;
    glCompileShader (shader) ;
    glGetShaderiv (shader, GL_COMPILE_STATUS, &compiled) ;
    if (!compiled) {
        shadererrors (shader) ;
        throw 3 ;
    }
    return shader ;
}
```

知乎 @WhyS0fAr

C风格的shader

## Vertex shader

attribute (顶点属性关键字只在vertex shader出现) vec3  
(3维向量) aVertexPosition(获取顶点位置)

vec4(aVertexPosition,1.0)定于一个4维向量只要3维向量后面  
跟一个数

varying (需要传递到fragment shader) high (定义计算精  
度) vec2 vNormol;

uniform(全局变量)

# Phong Shader in Assignment 0

- Vertex Shader

```
1  attribute vec3 aVertexPosition;  
2  attribute vec3 aNormalPosition;  
3  attribute vec2 aTextureCoord;  
4  
5  uniform mat4 uModelViewMatrix;  
6  uniform mat4 uProjectionMatrix;  
7  
8  varying highp vec2 vTextureCoord;  
9  varying highp vec3 vFragPos;  
10 varying highp vec3 vNormal;  
11  
12  
13 void main(void) {  
14  
15     vFragPos = aVertexPosition;  
16     vNormal = aNormalPosition;  
17  
18     gl_Position = uProjectionMatrix * uModelViewMatrix * vec4(aVertexPosition, 1.0);  
19  
20     vTextureCoord = aTextureCoord;  
21  
22 }
```

知乎 @WhyS0fAr

## Fragment Shader

color

=pow(texture2D(uSampler,vTextureCoord).rgb,vec3(2.2))  
(伽马矫正))

gl\_FragColor(输出颜色) =.....



- Fragment Shader

```
4  uniform sampler2D uSampler;
5  uniform vec3 uKd;
6  uniform vec3 uKs;
7  uniform vec3 uLightPos;
8  uniform vec3 uCameraPos;
9  uniform float uLightIntensity;
10 uniform int uTextureSample;
11
12 varying highp vec2 vTextureCoord;
13 varying highp vec3 vFragPos;
14 varying highp vec3 vNormal;
15
16 void main(void) {
17     vec3 color;
18     if (uTextureSample == 1) {
19         color = pow(texture2D(uSampler, vTextureCoord).rgb, vec3(2.2));
20     } else {
21         color = uKd;
22     }
```

知乎 @WhyS0fAr

- Fragment Shader (cont.)

```
24     vec3 ambient = 0.05 * color;
25
26     vec3 lightDir = normalize(uLightPos - vFragPos);
27     vec3 normal = normalize(vNormal);
28     float diff = max(dot(lightDir, normal), 0.0);
29     float light_atten_coff = uLightIntensity / length(uLightPos - vFragPos);
30     vec3 diffuse = diff * light_atten_coff * color;
31
32     vec3 viewDir = normalize(uCameraPos - vFragPos);
33     float spec = 0.0;
34     vec3 reflectDir = reflect(-lightDir, normal);
35     spec = pow(max(dot(viewDir, reflectDir), 0.0), 35.0);
36     vec3 specular = uKs * light_atten_coff * spec;
37
38     gl_FragColor = vec4(pow((ambient + diffuse + specular), vec3(1.0/2.2)), 1.0);
39
40 }
```

知乎 @WhyS0fAr

## Debug

早年debug只有NVIDIA Nsight能调试glsl，hlsl只能在软件上运行来debug；

Now

-Nsight Graphics, 跨平台, 但只支持NVIDIA。

-RenderDoc, 对显卡品牌没要求。

RGB 调试法

## **The Rendering Equation**

在games101中,我们整个path tracing的体系是建立在rendering equation上的,因为它是一个正确的用来描述光线传播的一个方法.

# The Rendering Equation

- Most important equation in rendering
  - Describing light transport

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{H^2} f_r(p, \omega_i \rightarrow \omega_o) L_i(p, \omega_i) \cos \theta_i d\omega_i$$



知乎 @WhyS0fAr

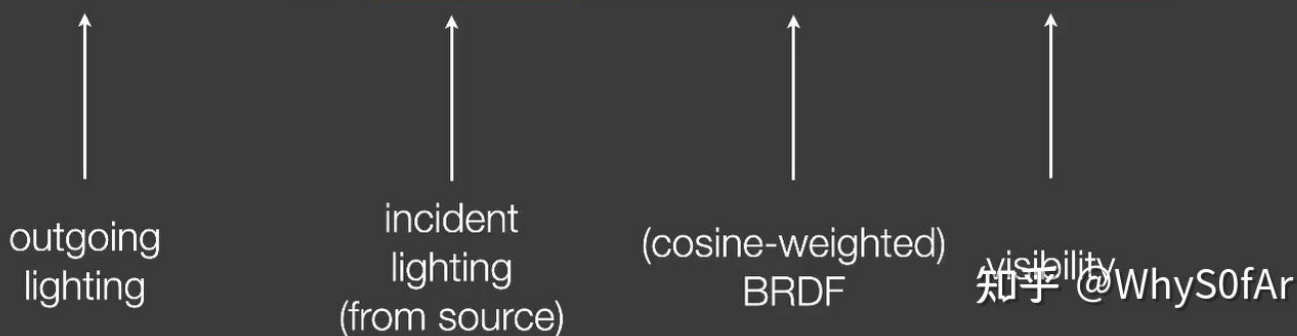
Rendering Equation 是一个正确的用来描述光线传播的等式，讲的是：

看到的任何一个点p反射到眼中的 radiance = 这个点p本身发出的radiance + 其他打到这个点的radiance \* brdf \* cos

- In real-time rendering (RTR)

- Visibility is often explicitly considered
- BRDF is often considered together with the cosine term

$$L_o(p, \omega_o) = \int_{\Omega^+} L_i(p, \omega_i) f_r(p, \omega_i, \omega_o) \cos \theta_i V(p, \omega_i) d\omega_i$$



我们在指brdf时候可能指原始brdf,也可能指cosine-weighted brdf

Visibility是在实时渲染中我们需要考虑到物体会不会被光源找到, 比如在一点p往一方向去看,我们知道这一方向有光源发出光线的,但是光线能否打到p上,因此引入Visibility 的概念。

real time rendering会显式的考虑visibility,他与原先的理解是等价的,之所以这么理解是为了能够更好理解环境光照,要比不写visibility项更加直观.

# Environment Lighting

- Representing incident lighting from all directions
  - Usually represented as a cube map or a sphere map (texture)
  - We'll introduce a new representation in this course



知乎 @WhyS0fAr

任何一方向过来的光的强度由一张图来决定,可以一个是cube box也可以是定义在球上的一张图,这两种表示都有不同的问题,在本课中会介绍一种新的表示,以八面体来表示.

此时只需要考虑说,从任何一个shading point往那个方向去,看这个visility是否可见,等于将实际上的光源和能否看到他,把这两项拆开考虑了,就很方便.

## 间接光照

光线会进行弹射, 形成间接光照, 将间接光照加加直接光照上就是全局光照, 实现全局光照就是要解决间接光照。



•p

**Direct illumination**

知乎 @WhyS0fA



•p

**One-bounce global illumination**

知乎 @WhyS0fA





我们可以看出,图像越亮越NB.

课堂答疑整理(此处是复制的祭曹大佬的整理)

[祭曹: GAMES202 Real-Time High Quality Rendering\\_\(高质量实时渲染\) 课程笔记Lecture 2: Recap of CG Basics](#)

- 1.问: 怎么定义哪些点连接成三角形? 答: 比如obj格式是先编号一系列点, 再定义面, 每个面都带着三个点的编号的下标。
- 2.问: 有无适合全局光照的管线? 答: 无, 但光线追踪渲染管线很完善。

3.问：纹理坐标怎么参数化的？答：可以理解成物体外面有个盒子，把盒子挤压到物体上，盒子上的uv是好确定的。

4.问：几个光源就需要几个shadowmap吗？光源也需要深度测试吗？答：是

5.问：opengl支持optix吗？答：可以，但不要指望shader能调用一个光线一个场景怎么运作。

6.问：纹理是不是是个buffer？答：opengl里两者定义不一样，但也可以这样理解，都是显存里的一块缓存区域。

7.问：一个pass就是一个frambuffer渲染一次吗？答：一个场景渲染一次

8.问：不同shader里定义里的变量会通用吗？答：不是