1 图形管线

我们已经大致学习了从二维图形的绘制, 再到三维图形的建模, 再到本章的三维场景的渲染. 现代的大型实时游戏场景中包含大量的三角形面片, 并且为了看起来流畅, 每秒渲染的帧数常常需要几十上百帧. 为了高效完成图形渲染的工作, 人们对渲染的流程进行了长时间的优化和演进, 最终形成了**图形管线 (Graphic Pipeline)** 的概念.

定义 1.1 图形管线 **图形管线**是在 GPU/图形硬件层面上, 将顶点与图元的输入经过一系列确定的硬件/着色器阶段 (顶点处理, 图元装配, 裁剪/投影, 光栅化, 片段处理, 像素测试与输出合并等), 以产生最终帧缓冲中像素值的有序数据处理流水线.

通常, 我们所说的**渲染管线 (Rendering Pipeline)** 大致与图形管线相同.

1.1 渲染管线的一般过程

通常, 渲染管线可以分为以下几个步骤, 每一个步骤接受的输入都是上一步骤的输出.

(1) 应用 (Application)

- 目的: 构建和提交需要渲染的几何数据和渲染状态.
- 输出: 由需要渲染的 3D 应用 (例如游戏) 给出, 包括模型数据, 纹理数据, 相机参数, 光源信息等等[1].

(2) 顶点处理 (Vertex Processing)

- 目的: 将模型变换到裁剪空间[2]; 然后计算顶点处的各种属性 (包括法线, UV 坐标, 颜色等).
- 输出: 顶点在裁剪空间中的位置及其它属性.

(3) 三角形处理 (Triangle Processing)

- 目的: 将顶点组装成三角形图元, 并进行裁剪和投影变换.
- 过程:
  - **a. 图元装配**: 将顶点按预定的模式组装成三角形图元.
  - **b. 背面剔除**: 基于顶点顺序与法线方向的关系删除不可见的三角形.
  - **c. 裁剪**: 将超出视锥的三角形裁剪裁剪成视锥内的部分[3].
  - **d. 透视除法与视口变换**: 将裁剪空间中的顶点通过透视除法转换到 NDC 空间, 然后通过视口变换映射到屏幕空间.

[1] 通常, 在渲染过程中采取的加速算法也由 CPU 执行, 因此包含在此部分内.

[2] 通过我们在几何变换中所学的投影变换的方法转换到裁剪空间中. 裁剪空间中顶点以齐次坐标的形式表示, 并且第四个分量 w 尚未进行归一化. 对裁剪空间中顶点坐标做下述变换

$$\begin{bmatrix} x & y & z & w \end{bmatrix}^{\mathrm{t}} \longrightarrow \begin{bmatrix} \frac{x}{w} & \frac{y}{w} & \frac{z}{w} \end{bmatrix}^{\mathrm{t}}$$

即归一化并除去第四个分量后才能转换成规范化坐标系, 即 **NDC(Normalized Device Coordinates) 空间**. 这一步被称作**透视除法**, 是下一步三角形处理中的操作.

[3] 这一步就是裁剪空间这一名称的由来.

- 输出: 屏幕空间中的三角形片元 (包含各种属性).

(4) 光栅化 (Rasterization)

- 目的: 将三角形片元转换为屏幕的片元 (Fragments, 可以理解为采样点), 并生成用于插值的片元数据[4].
- 输出: 片元流 (Fragment Stream), 其中每个片元包含插值后的属性 (颜色, 法线, UV 坐标, 深度等).

(5) 片元处理 (Fragment Processing)

- 目的: 通过给定的着色器, 以及片元的各种数据 (包括纹理坐标, 法线, 顶点颜色, 深度等) 计算片元的最终颜色值.
- 输出: 每个片元的最终颜色值和深度值.

(6) 帧缓冲操作 (Framebuffer Operations)

- 目的: 将片元的颜色和深度值与帧缓冲中的现有值进行测试和合并, 以生成最终的像素值.
- 过程:
  a. **深度测试**: 比较片元的深度值与帧缓冲中对应像素的深度值, 以确定片元是否可见.
  b. **模板测试**: 基于模板缓冲的值决定片元是否应被绘制. 通常用于复杂遮罩, 轮廓等高级效果的控制.
  c. **混合**: 将片元的颜色值与帧缓冲中现有的颜色值进行混合 (如果需要), 以实现透明度等效果.

(7) 显示 (Display)

- 目的: 将帧缓冲中的最终像素值传输到显示设备 (例如监视器) 以进行可视化.
- 输出: 显示设备上呈现的最终图像.

1.2 图形 API

为了实现对图形管线的控制和使用, 现代计算机图形学中引入了**图形 API(Graphic API)** 的概念.

定义 1.2 图形 API **图形 API** 是指应用程序与图形硬件之间的抽象接口, 它通过一组函数或指令集, 使开发者能够控制 GPU 执行图形渲染, 计算和资源管理等操作, 而无需直接编写底层驱动代码.
常见图形 API 包括 OpenGL, Vulkan, DirectX, Metal 等.

上面介绍的渲染管线仅为通用的大致结构, 各个图形 API 对管线的具体实现过程是不同的. 以 OpenGL 为例, 其渲染管线如下所示.

[4] 一些超采样算法, 例如 MSAA, 需要在此时进行覆盖测试; 此外, 片元的插值需要经过透视矫正.

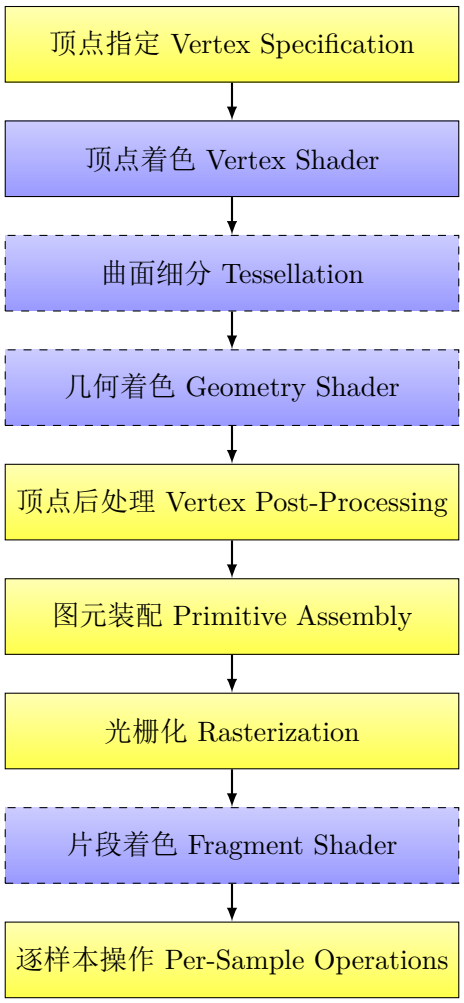

图 1: OpenGL 渲染管线示意图

上述流程图中仅有标蓝色框的部分是可编程的, 其余步骤均在硬件驱动中实现, 仅能通过一些选项调整参数. 大部分情况下需要手动编写的仅有**顶点着色器**.