第九章

三维图形绘制流水线

一、创建一个基本D3D程序

- 创建一个Window窗口。
  - 注册WNDCLASS类: RegisterClass
  - CreateWindow(…)
  - MsgProc(HWND hwnd, UINT msg, WPARRAM wParam, LPARAm lParam)

- 初始化Direct3D程序
  - 创建Direct3D对象: Direct3DCreate9()
  - 查询显卡的显示模式: GetAdapterDisplayMode()
  - 创建Direct3D设备对象: CreateDevice(…)

- 处理消息循环

```
MSG msg:
While(GetMessage(&msg, NULL,0,0))
{
    TranslateMessage(&msg)
    DispatchMessage(&msg)
}
```

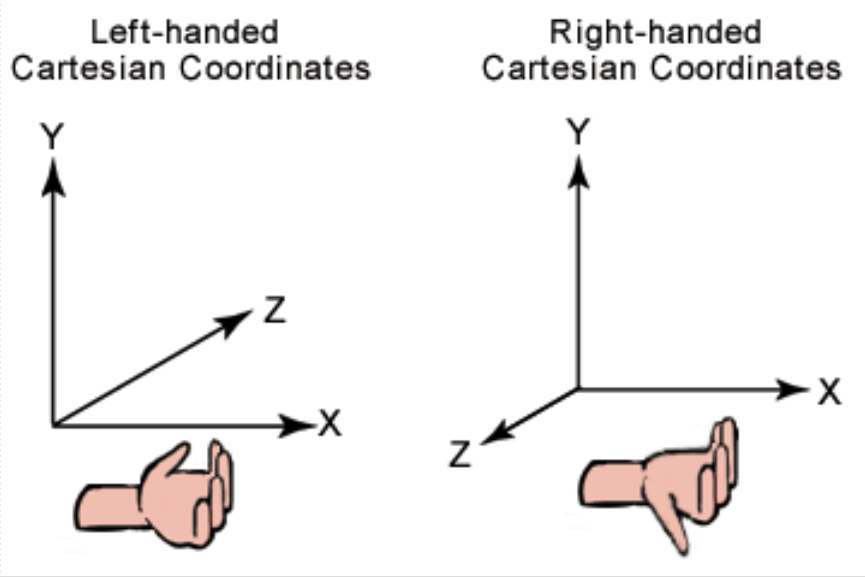
- 图形显示
  - 清屏: IDirect3DDevice9::Clear()
  - BeginScene()和EndScene():不嵌套
  - 更新缓冲: IDirect3DDevice9::Present()
- 结束Direct3D程序
  - Cleanup()
  - UnregisterClass()

D3D初始化

- 1. 初始化主显示窗口和Direct3D
- 2. 调用Setup进行程序的准备工作
- 3. 使用Display函数作为参数进入消息循环
- 4. 清除应用程序最后释放IDirecte3DDevice9对象

基本几何模型显示

- 3D坐标系

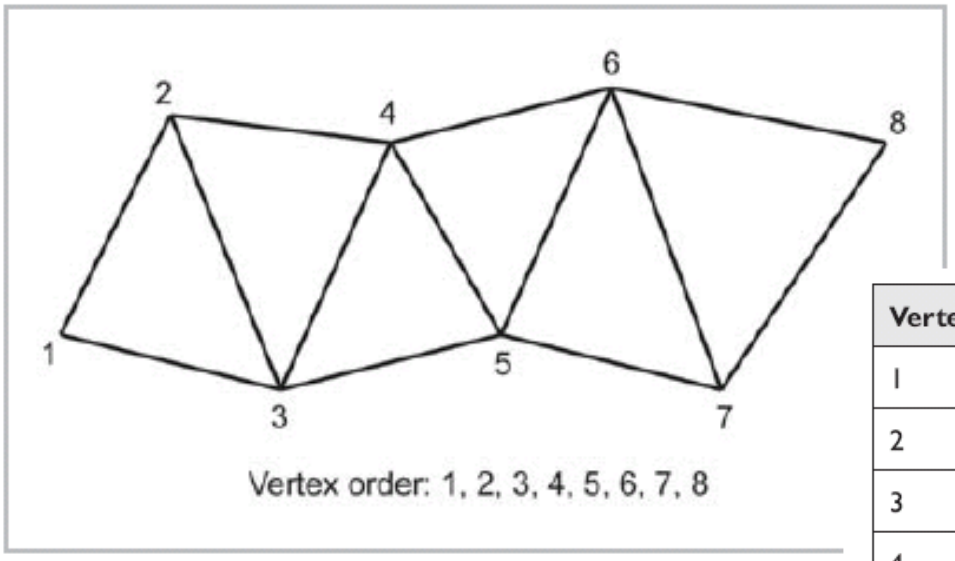


Direct3D uses left-handed, OpenGL (and most math) uses right-handed

图元绘制

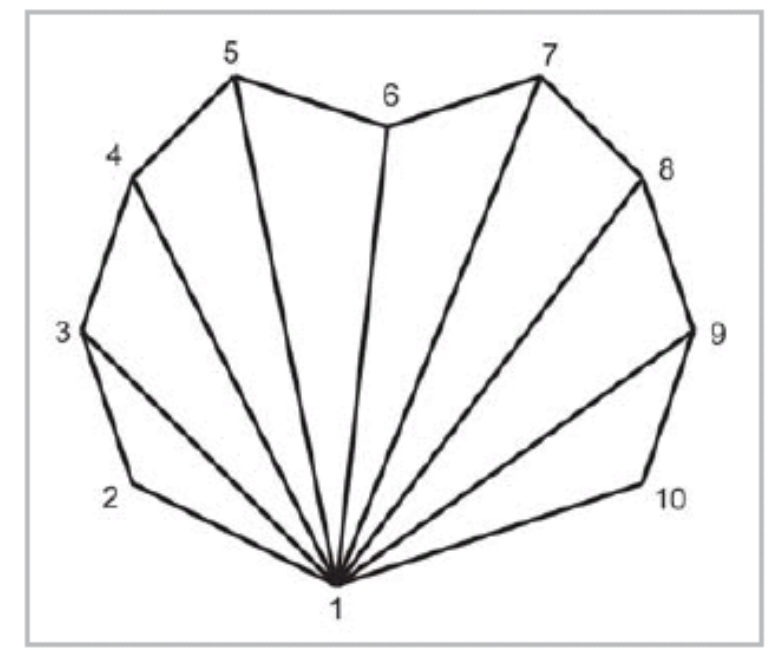
- 定义点的集合
- DrawPrimitive():
- 六种基本图元: D3DPT_POINTLIST
  D3DPT_LINELIST
  D3DPT_LINESTRIP
  D3DPT_TRIANGLELIST
  D3DPT_TRIANGLESTRIP
  D3DPT_TRIANGLEFAN

A triangle strip



Vertex	Triangle	Vertex Ordering
1	(none)	
2	(none)	
3	1,2,3	Clockwise
4	2,3,4	Counterclockwise
5	3,4,5	Clockwise
6	4,5,6	Counterclockwise
7	5,6,7	Clockwise
8	6,7,8	Counterclockwise

Triangle Fans



- D3DPT_LINELIST: 一组线段的集合;
- D3DPT_LINESTRIP:首尾相接不间断的点表示一组线段;
- D3DPT_TRIANGLELIST: 孤立的一组三角形的集合;(背向剔除自动启用)
- D3DPT_TRIANGLESTRIP:首尾相接不间断的一组三角形集合;(偶数自动翻转)
- D3DPT_TRIANGLEFAN:每一个三角形的前两个顶点分别是第一个三角形的第一个顶点和前一个三角形的最后一个顶点.

•1.顶点缓冲区

- 一个顶点缓存是一块连续的存储了顶点数据的内存
- 我们使用顶点和索引缓存保存我们的数据是因为它们能被放置在显存中。渲染显存中的数据要比渲染系统内存中的数据快的多;
- 在代码中，一个顶点缓存是通过IDirect3DVertexBuffer9接口来定义的。
- 可以在顶点缓冲区内对顶点进行坐标变换,光照,裁减
- 利用顶点缓冲区内的顶点构成基本图元输出

创建一个顶点和索引缓存

```
HRESULT IDirect3DDevice9::CreateVertexBuffer(
    UINT Length,
    DWORD Usage,
    DWORD FVF,
    D3DPOOL Pool
    IDirect3DVertexBuffer9** ppVertexBuffer,
    HANDLE* pSharedHandle
);
```

- Length —— 分配给缓存的字节大小。假如想得到一个能存储8个顶点的顶点缓存，那么我们就要在顶点结构中设置这个参数为 8 * sizeof (Vertex) 。
- Usage —— 指定关于怎样使用缓存的额外信息。这个值可以是0，没有标记，或者是下面标记的一个或多个的组合：D3DUSAGE_DYNAMIC——设置这个参数可以使缓存是动态的。在下一页说明静态和动态缓存。
- D3DUSAGE_POINTS——这个参数指定缓存存储原始点。原始点将在第14章粒子系统中介绍。这个参数仅仅用在顶点缓冲中。
- D3DUSAGE_SOFTWAREPROCESSING——使用软件顶点处理
- D3DUSAGE_WRITEONLY——指定应用程序只能写缓存。它允许驱动程序分配最适合的内存地址作为写缓存。注意如果从创建好的这种缓存中读数据，将会返回错误信息。
- FVF —— 存储在缓存中的顶点格式
- Pool —— 缓存放置在哪一个内存池中
- ppVertexBuffer ——返回创建好的顶点缓存的指针。
- pSharedHandle ——没有使用；设置为0。

访问缓冲内存

我们通过一个指针获得缓存数据必须使用Lock方法。当我们访问完缓存后必须对它解锁。

```
HRESULT IDirect3DVertexBuffer9::Lock(
    UINT OffsetToLock,
    UINT SizeToLock,
    BYTE** ppbData,
    DWORD Flags
);
HRESULT IDirect3DIndexBuffer9::Lock(
    UINT OffsetToLock,
    UINT SizeToLock,
    BYTE** ppbData,
    DWORD Flags
);
```

- OffsetToLock —— 偏移量，以字节为单位，从缓存开始位置到锁定开始位置的距离。如图3.1。
- SizeToLock —— 锁定的字节数。
- ppbData —— 一个指向锁定内存开始位置的指针。
- Flags —— 标记描述怎样锁定内存。

```
Vertex* vertices;
_vb->Lock(0, 0, (void**)&vertices, 0); // lock the entire buffer
vertices[0] = Vertex(-1.0f, 0.0f, 2.0f); // write vertices to
vertices[1] = Vertex( 0.0f, 1.0f, 2.0f); // the buffer
vertices[2] = Vertex( 1.0f, 0.0f, 2.0f);
_vb->Unlock(); // unlock when you' re done accessing the buffer
```

2.渲染状态

Direct3D提供多种渲染状态，它影响几何物体怎样被渲染。

```
HRESULT IDirect3DDevice9::SetRenderState(
    D3DRENDERSTATETYPE State, // the state to change
    DWORD Value // value of the new state
);
```

```
_device->SetRenderState(D3DRS_FILLMODE, D3DFILL_WIREFRAME);
```

3.绘制准备

- 一旦我们创建好一个顶点缓存以及一个索引缓存（可选的）后：
  - 1、 设置资源流。设置资源流与一个顶点缓存挂钩，此流就是一个流入渲染管线的几何信息的流。

```
HRESULT IDirect3DDevice9::SetStreamSource(
    UINT StreamNumber,
    IDirect3DVertexBuffer9* pStreamData,
    UINT OffsetInBytes,
    UINT Stride
);
```

- StreamNumber——确定我们的顶点缓存与哪一个资源流挂钩。一般总是使用0号流。
- pStreamData——一个指向我们想与流挂钩的那个顶点缓存的指针。
- OffsetInBytes——相对流开始处的偏移量。以字节为单位，它指定被填入渲染管道的顶点数据的开始位置。通过检查D3DCAPS9结构中的D3DDEVCAPS2_STREAMOFFSET标志，假如你的设备支持，那么这个参数就有一些非0值。
- Stride——我们在顶点缓存中操作的每个部分的流的字节大小。

例如，假设vb是一个已经填充了顶点信息的顶点缓存：

```
_device->SetStreamSource( 0, vb, 0, sizeof( Vertex ) );
```

- 设置顶点格式。在这里我们指定后面用来绘图调用的顶点的顶点格式。

```
_device->SetFVF( D3DFVF_XYZ | D3DFVF_DIFFUSE | D3DFVF_TEX1 );
```

4.用顶点/索引缓存绘制

```
HRESULT IDirect3DDevice9::DrawPrimitive(
    D3DPRIMITIVETYPE PrimitiveType,
    UINT StartVertex,
    UINT PrimitiveCount
);
```

- PrimitiveType——我们绘制的图元类型。比如，我们能绘制点和线以及三角形。以后我们使用三角形，用D3DPT_TRIANGLELIST参数。
- StartVertex——索引到在顶点流中的一个元素。设置渲染顶点中的开始点。这个参数给予我们一定的机动性去绘制一个顶点缓存中的某部分。
- PrimitiveCount——绘制图元的个数。

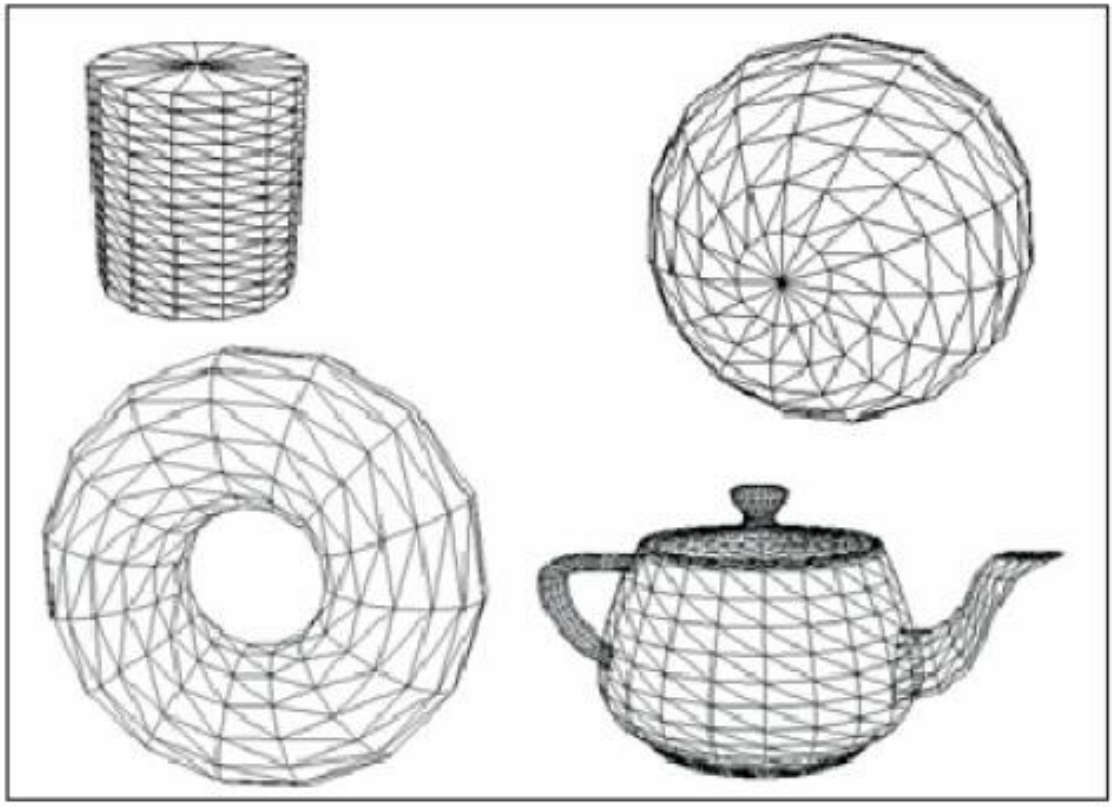
开始/结束场景

所有绘制方法都必须在IDirect3DDevice9::BeginScene和IDirect3DDevice9::EndScene方法之间被调用

```
_device->BeginScene();
    _device->DrawPrimitive(...);
_device->EndScene();
```

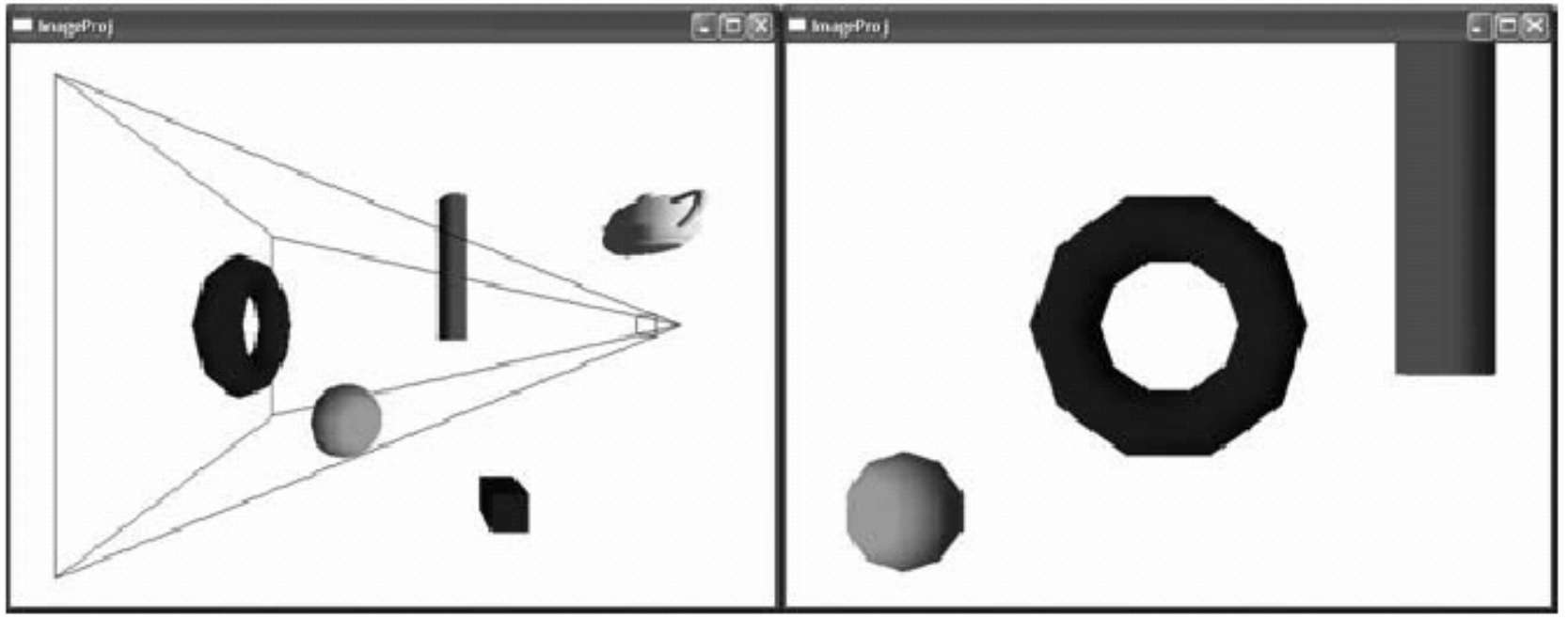
5.D3DX几何物体

- D3DX库已经为我们提供了一些方法来产生简单3D物体的网格数据。
- D3DX库提供如下6种网格生成函数。
  - D3DXCreateBox
  - D3DXCreateSphere
  - D3DXCreateCylinder
  - D3DXCreateTeapot
  - D3DXCreatePolygon
  - D3DXCreateTorus



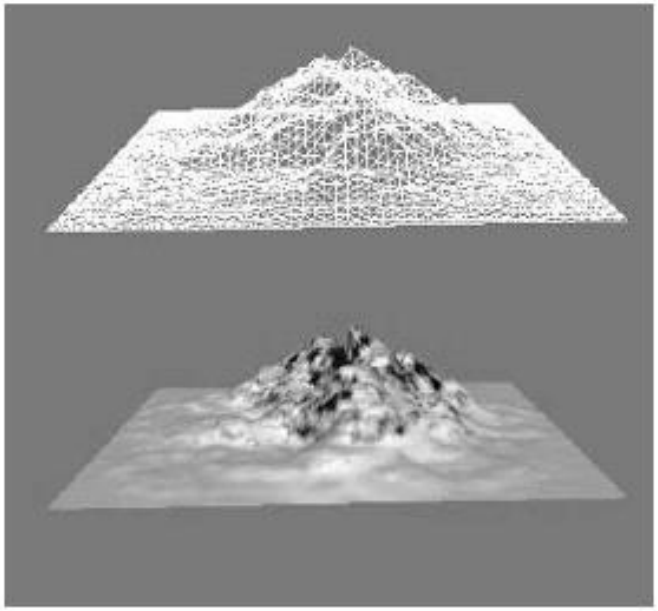
二、三维图形的绘制流水线

- 渲染管道

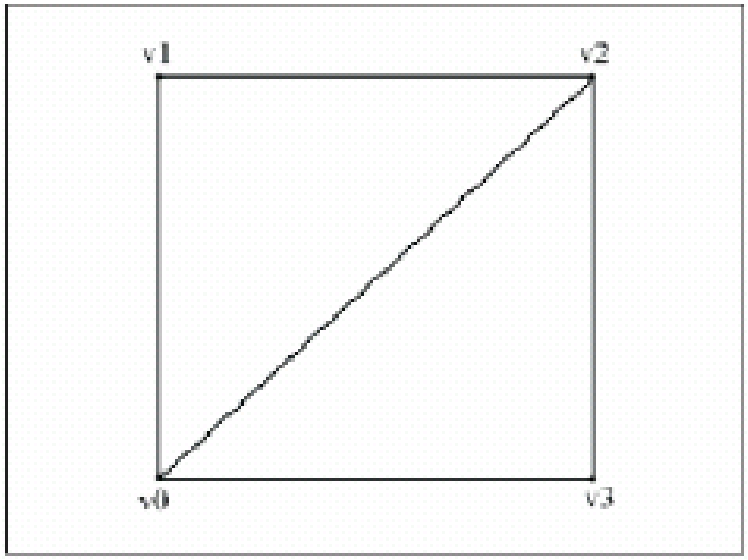


三维场景的表现模型

- 最常见的就是Mesh(三角网格)



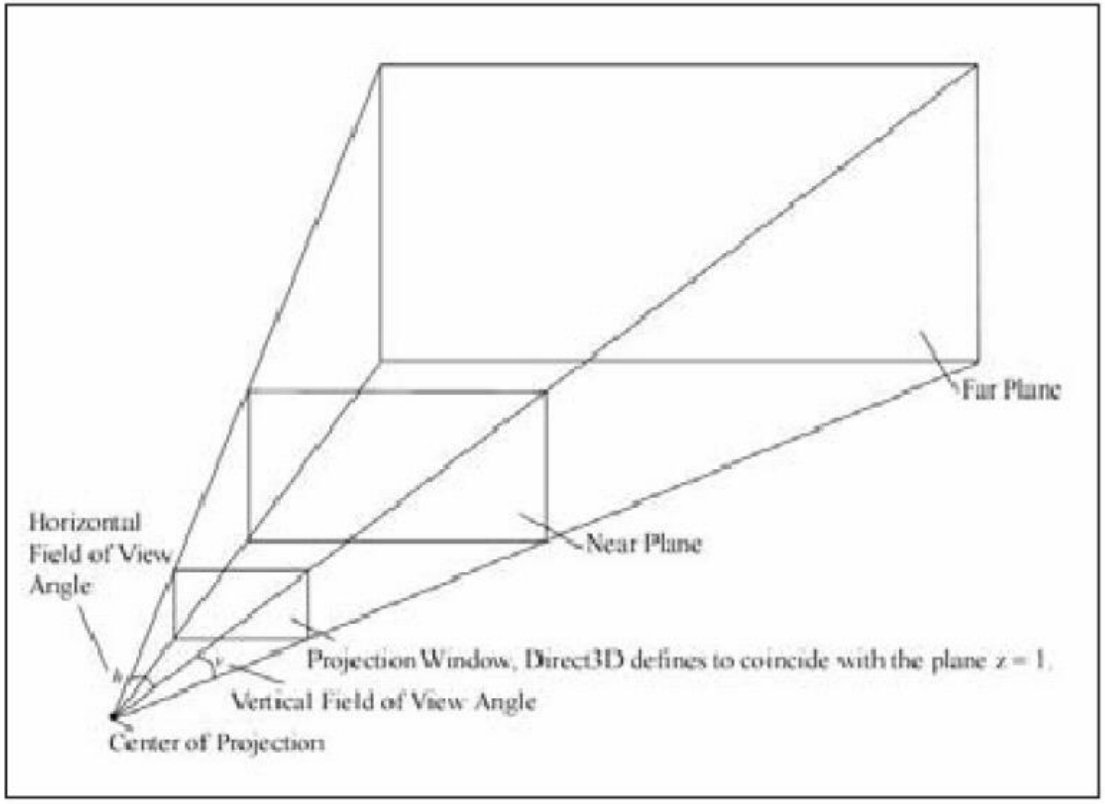
三角形



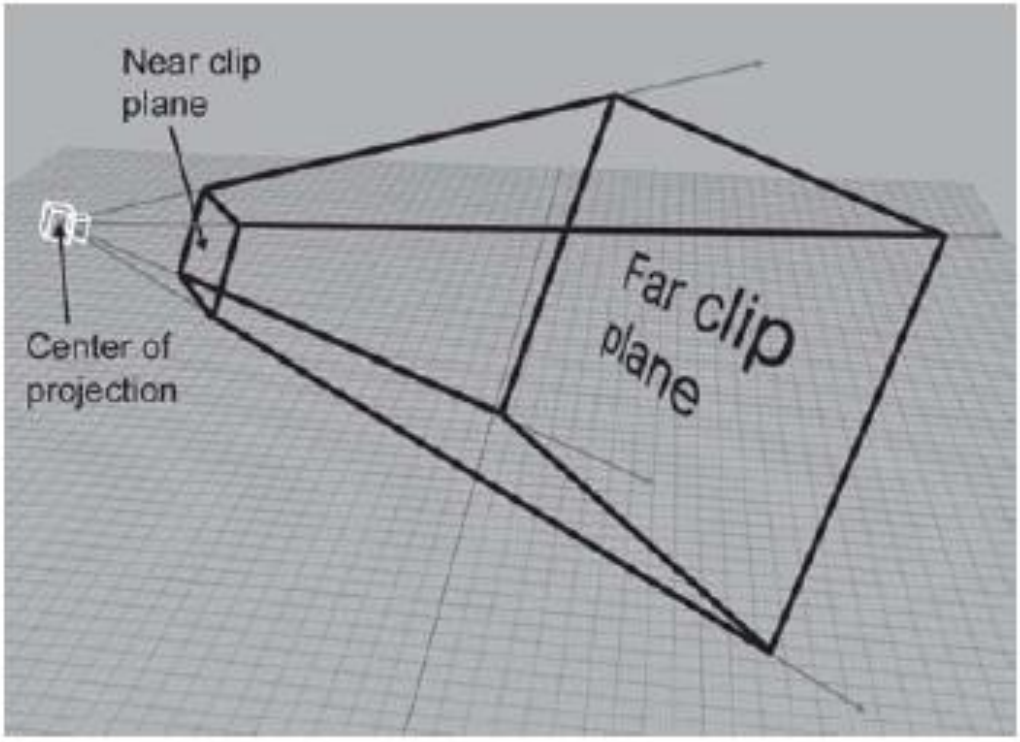

```
Vertex rect[6] = {v0, v1, v2, // triangle0
                  v0, v2, v3}; // triangle1
```

注意：指定三角形顶点的顺序是很重要的，将会按一定顺序环绕排列，不然会因为背向剔除看不见。

虚拟摄相机

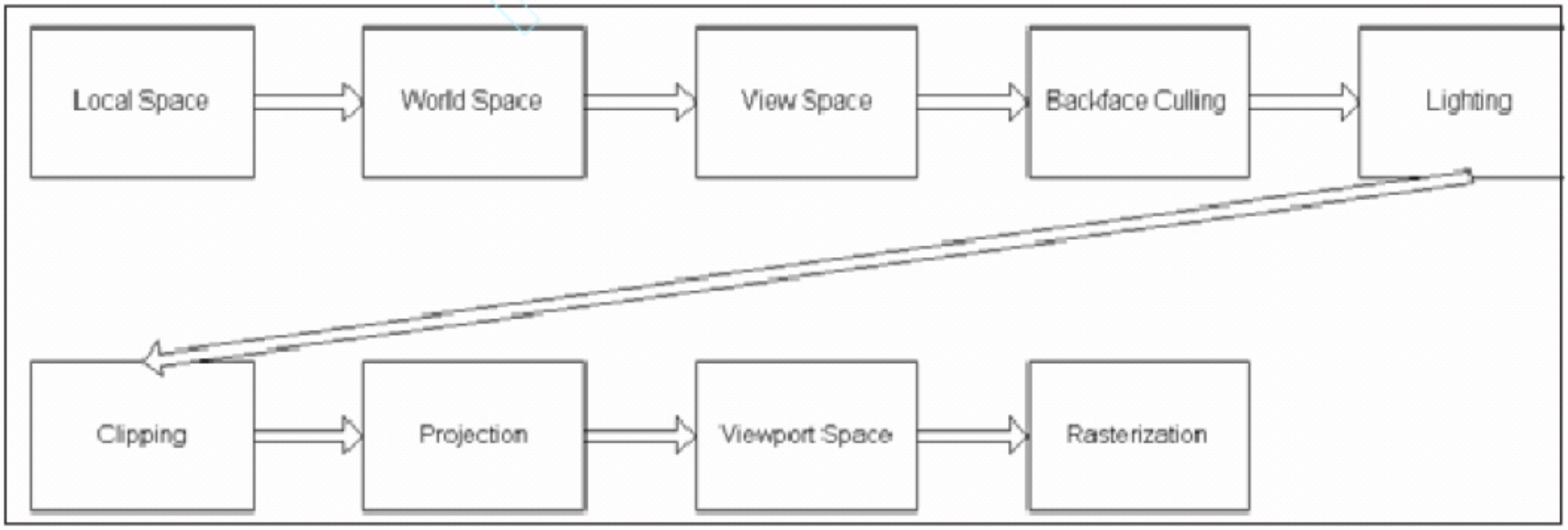


摄相机确定3D世界中的哪部分是可见的因而需要将哪部分转换为2D图形

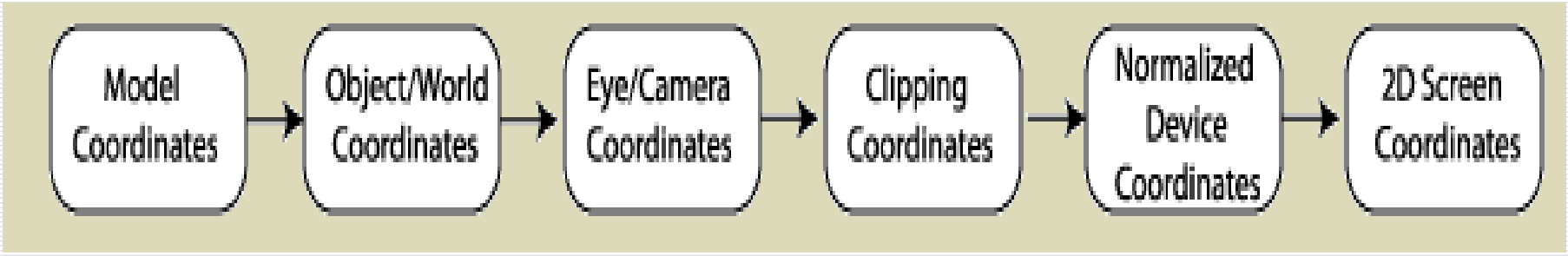

Near clip
plane
Far clip
plane
Center of
projection

渲染管道

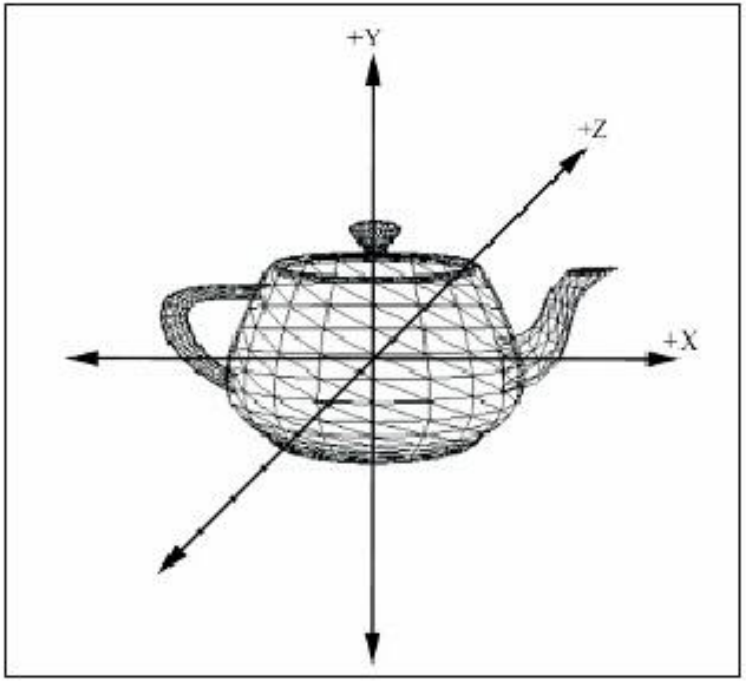
- 一旦我们描述几何学上的3D场景和设置了虚拟摄相机，我们要把这个场景转换成2D图象显示在显示器上。



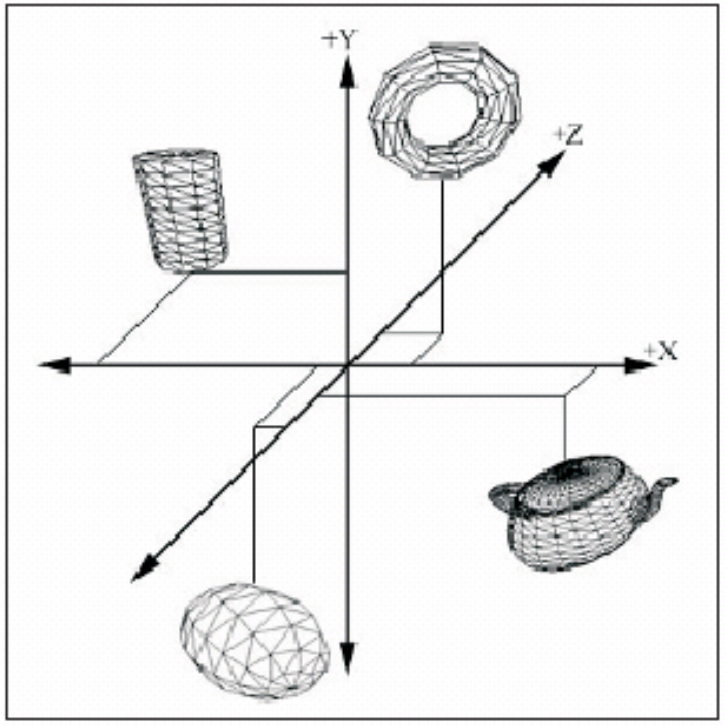
绘制流水线


Model Coordinates
Object/World Coordinates
Eye/Camera Coordinates
Clipping Coordinates
Normalized Device Coordinates
2D Screen Coordinates

1.自身坐标系（**Local Space**）



2.世界坐标系（**World Space**）

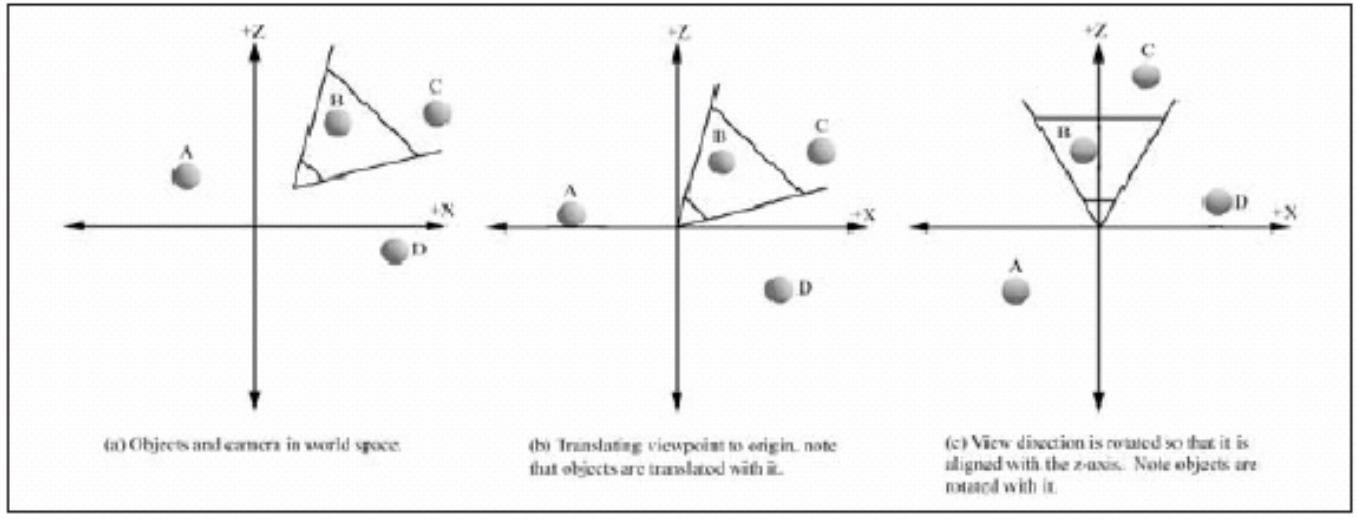


变换通常是用平移、旋转、缩放操作来设置模型在世界坐标系中的位置、大小、方向。

```
Device->SetTransform(D3DTS_WORLD, &worldMatrix);
```

3.视图坐标系（**View Space**）

- 在世界坐标系中当摄相机是任意放置和定向时，投影和其它一些操作会变得困难或低效。为了使事情变得更简单，我们将摄相机平移变换到世界坐标系的源点并把它的方向旋转至朝向Z轴的正方向，当然，世界坐标系中的所有物体都将随着摄相机的变换而做相同的变换。这个变换就叫做视图坐标系变换.

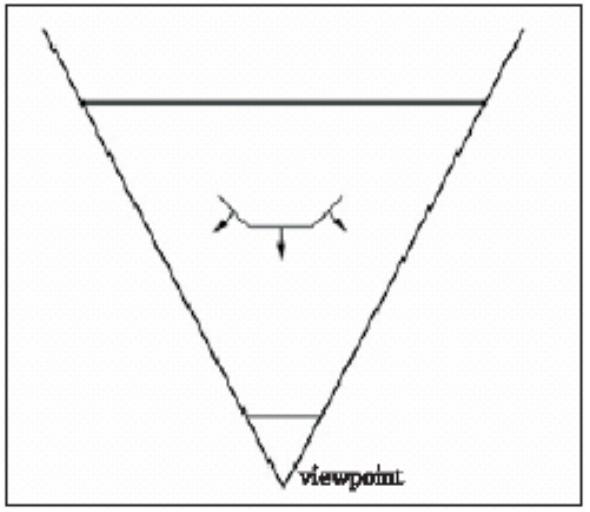


```
D3DXMATRIX *D3DXMatrixLookAtLH(
    D3DXMATRIX* pOut, // pointer to receive resulting view matrix
    CONST D3DXVECTOR3* pEye, // position of camera in world
    CONST D3DXVECTOR3* pAt, // point camera is looking at in world
    CONST D3DXVECTOR3* pUp // the world's up vector - (0, 1, 0)
);
```

```
Device->SetTransform(D3DTS_VIEW, &V);
```

4.背面剔除（**Backface Culling**）

- Direct3D将通过拣选（即删除多余的处理过程）背面多边形来提高效率，这种方法就叫背面剔除.
- Direct3D中默认顶点以顺时针方向（在观察坐标系中）形成的三角形为正面，以逆时针方向形成的三角形为背面。



- 如果我们不想使用默认的拣选状态，我们可以通过改变D3DRS_CULLMODE来改变渲染状态：

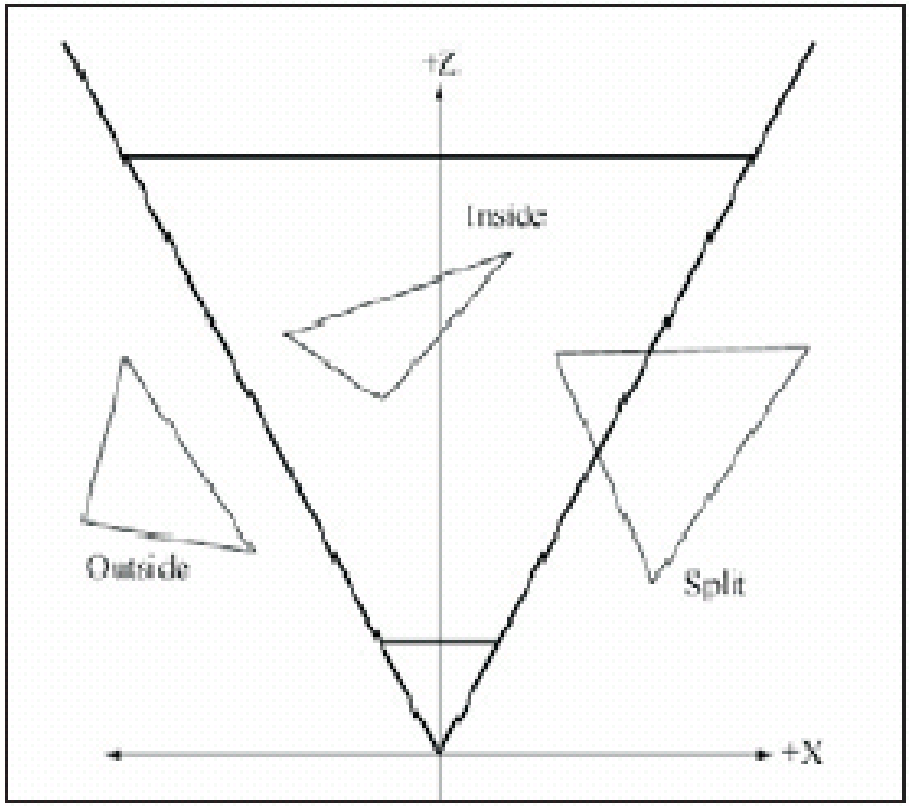
```
Device->SetRenderState(D3DRS_CULLMODE, Value);
```

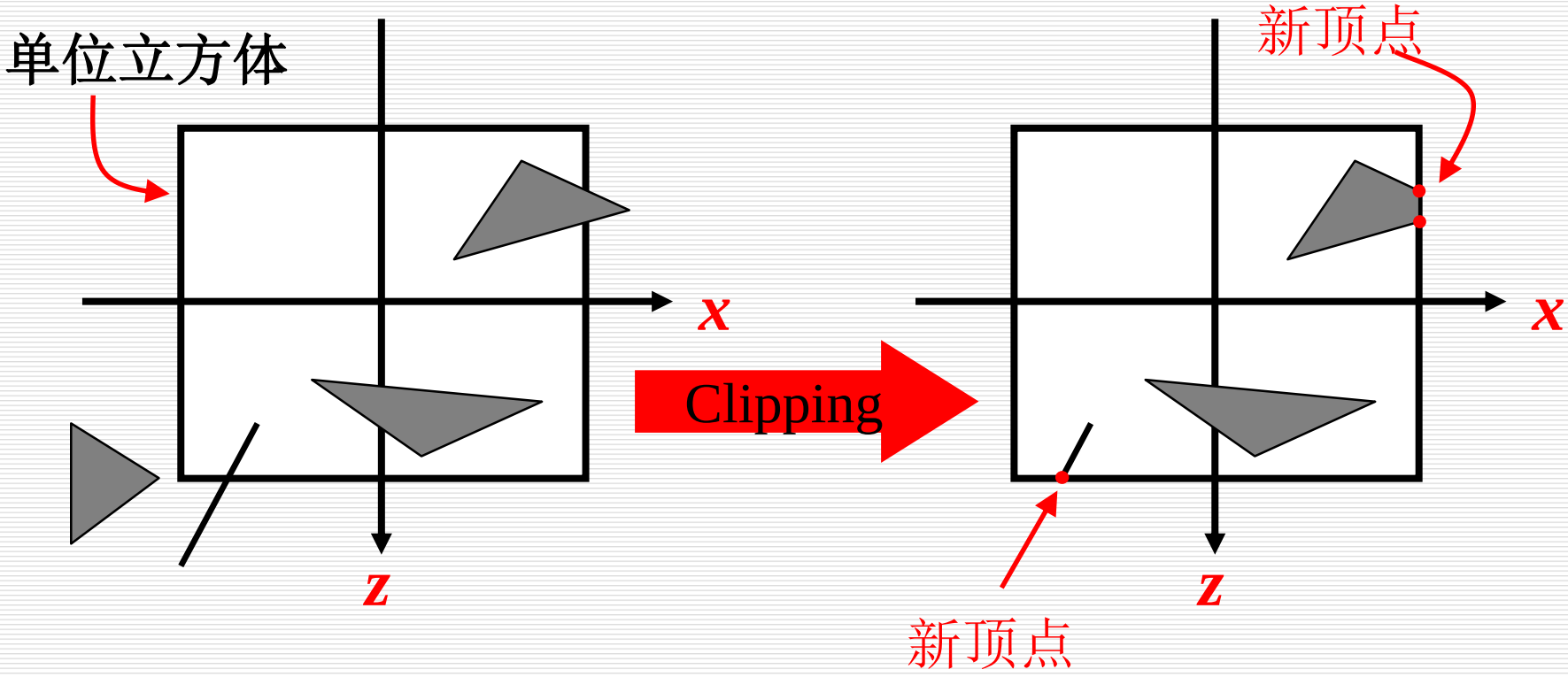
- Value可以是如下一个值：
  - D3DCULL_NONE——完全不使用背面拣选
  - D3DCULL_CW——拣选顺时针环绕的三角形
  - D3DCULL_CCW——逆时针方向环绕的三角形会被拣选，这是默认值

- 5光源（**Lighting**）
  - 视图坐标系中光源给物体施加的光照大大增加了场景中物体的真实性

6裁剪（**Clipping**）

- 我们拣选那些超出了可视体范围的几何图形的过程就叫做裁剪。这会出现三种情况：
  - 完全包含——三角形完全在可视体内，这会保持不变，并进入下一级
  - 完全在外——三角形完全在可视体外部，这将被拣选
  - 部分在内（部分在外）——三角形一部分在可视体内，一部分在可视体外，则三角形将被分成两部分，可视体内的部分被保留，可视体之外的则被拣选

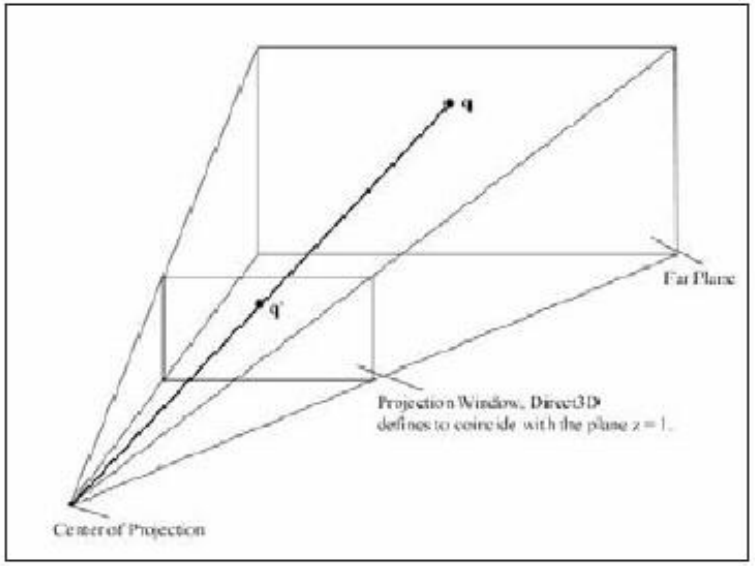

+Z
Inside
Outside
Split
+X


单位立方体
x
z
Clipping
新顶点
x
z
新顶点

7 投影

- 3D场景转化为2D图像表示。这种从n维转换成n-1维的过程就叫做投影。

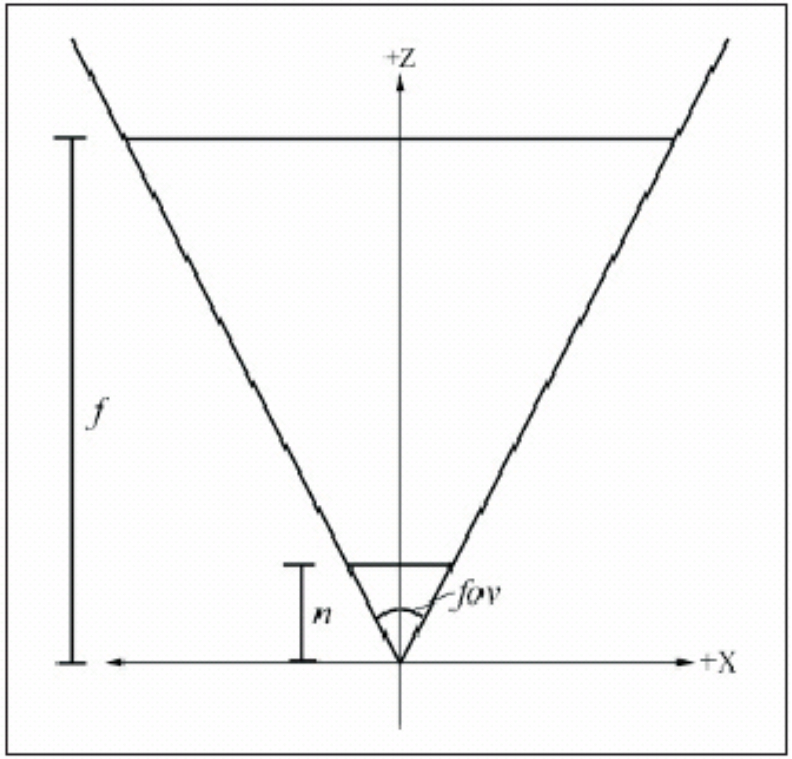
- 最常用的是透视投影.
  - 可以使离摄相机越远的物体投影到屏幕上后就越小，这可以使我们把3D场景更真实的转化为2D图像。



- 投影变换的实质就是定义可视体并将可视体内的几何图形投影到投影窗口上去。

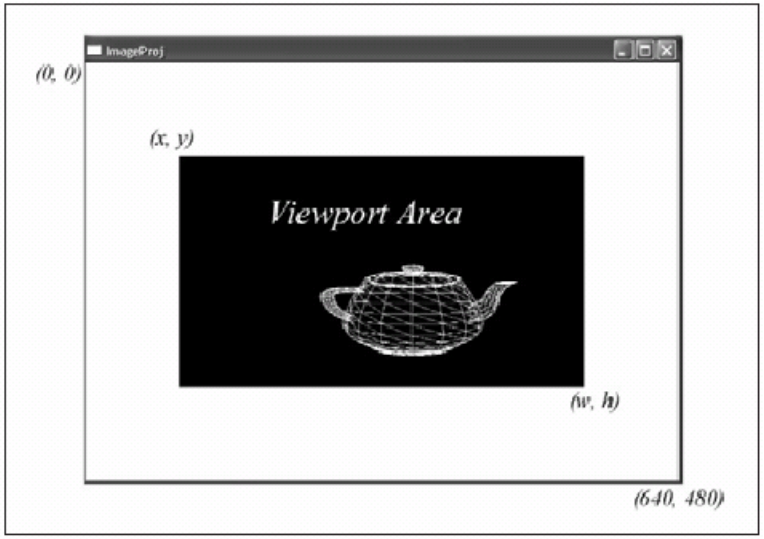
```
D3DXMATRIX *D3DXMatrixPerspectiveFovLH(
    D3DXMATRIX* pOut, // returns projection matrix
    FLOAT fovY, // vertical field of view angle in radians
    FLOAT Aspect, // aspect ratio = width / height
    FLOAT zn, // distance to near plane
    FLOAT zf // distance to far plane
);
```

```
D3DXMATRIX proj;
D3DXMatrixPerspectiveFovLH(
    &proj, PI * 0.5f, (float)width / (float)height, 1.0, 1000.0f);
Device->SetTransform(D3DTS_PROJECTION, &proj);
```



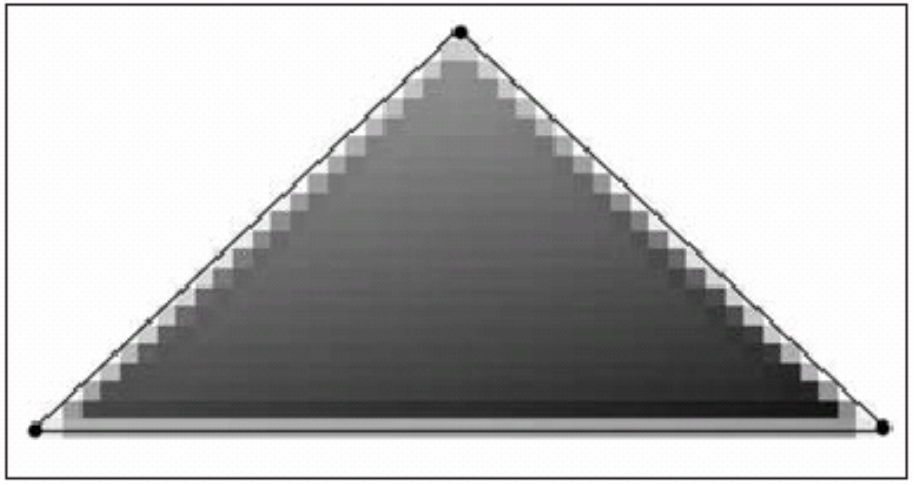
8 视口变换（**Viewport Transform**）

- 视口变换主要是转换投影窗口到显示屏幕上。通常一个游戏的视口就是整个显示屏，但是当我们以窗口模式运行的时候，也有可能只占屏幕的一部分或在客户区内.



9 光栅化（**Rasterization**）

- 在把三角形每个顶点转换到屏幕上以后，我们就画了一个2D三角形。光栅化是计算需要显示的每个三角形中每个点颜色值.



- 通过硬件图形处理来完成.

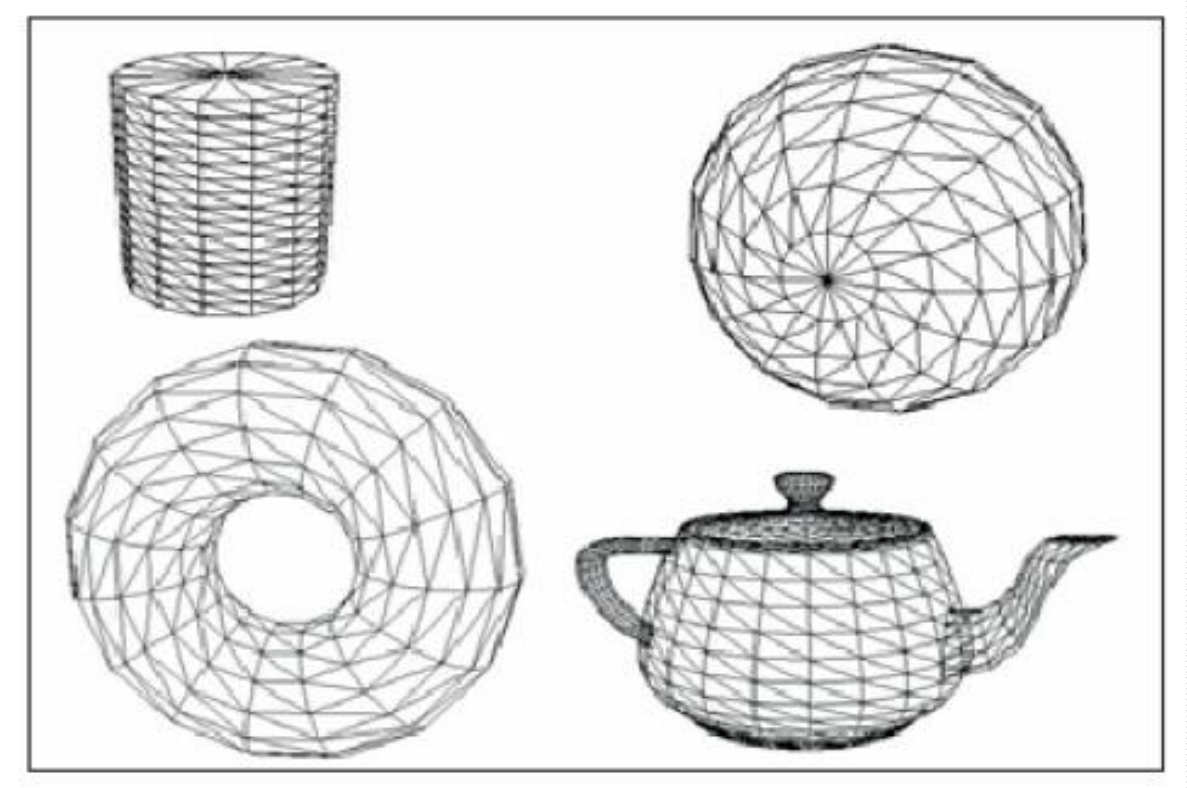
D3D中的关键Pipline

- Step 1 - Defining the World ransformation Matrix
- Step 2 - Defining the View Transformation Matrix
- Step 3 - Defining the Projection Transformation Matrix

- Direct3D applies the matrices to the scene in the following order:
  - World
  - View
  - Projection

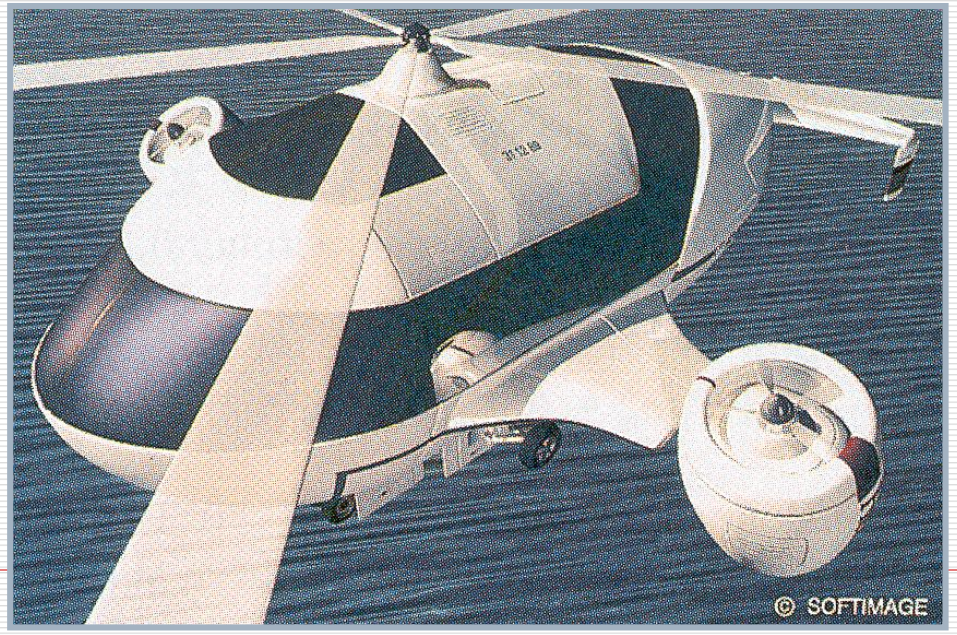
- http://www.willamette.edu/~gorr/classes/GeneralGraphics/Pipeline/geometry.htm
- http://www.cs.princeton.edu/courses/archive/fall99/cs426/lectures/transform/

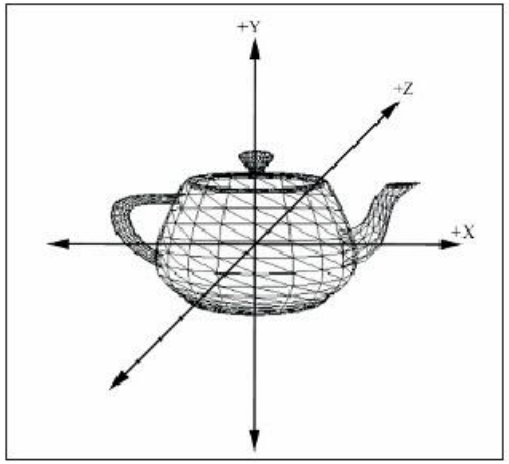
绘制复杂物体的过程



绘制复杂物体的过程

- 怎样将现实中的一个物体，比如，一只花瓶，一个足球，甚至一架大的战斗机，在电脑屏幕上显示呢？





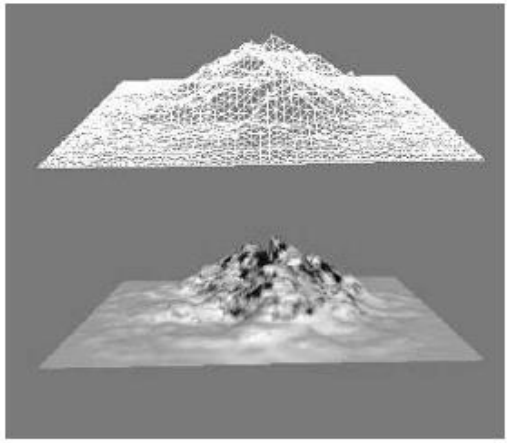
1. 3D建模

- 先把该物体放在一个虚拟的三维坐标系中，该坐标称为局部坐标系(Local Space), 一般以物体的中心作为坐标原点，采用左手坐标系。

  然后，对坐标系中的物体进行点采样, 这些采样点按一定顺序连接成为一系列的小平面, 这些小平面称为图元(Primitive), 3D引擎会处理每一个图元，称为一个独立的渲染单位。这样取样后的物体看起来像是由许许多多的三角形，四边形或五边形组成的，就像网一样，我们称为一个网格(Mesh).

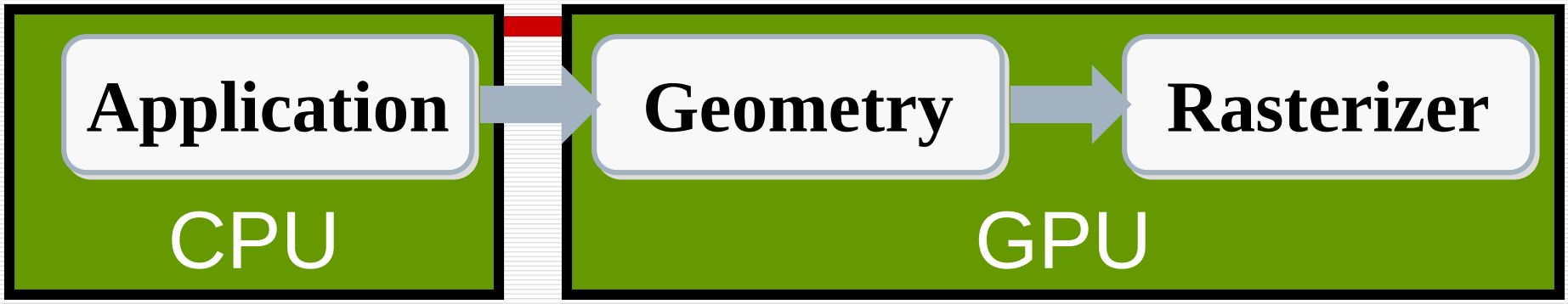
- 这个采样过程又可称为物体的3D建模, 当然现在都有功能非常强大的3D建模工具，例如3D Max。

2. 绘制



- 我们纪录这些顶点数据和连线情况到一个文件中，3D引擎读取这些数据，依次渲染每一个图元，就能在显示屏幕上再现物体。
- 在D3D中，纪录这些顶点数据和连线情况的文件称为X文件(X File)。它是以X作为文件名后缀的。
- 也有其他类型的模型文件，主要保存模型的顶点数据和面片数据。

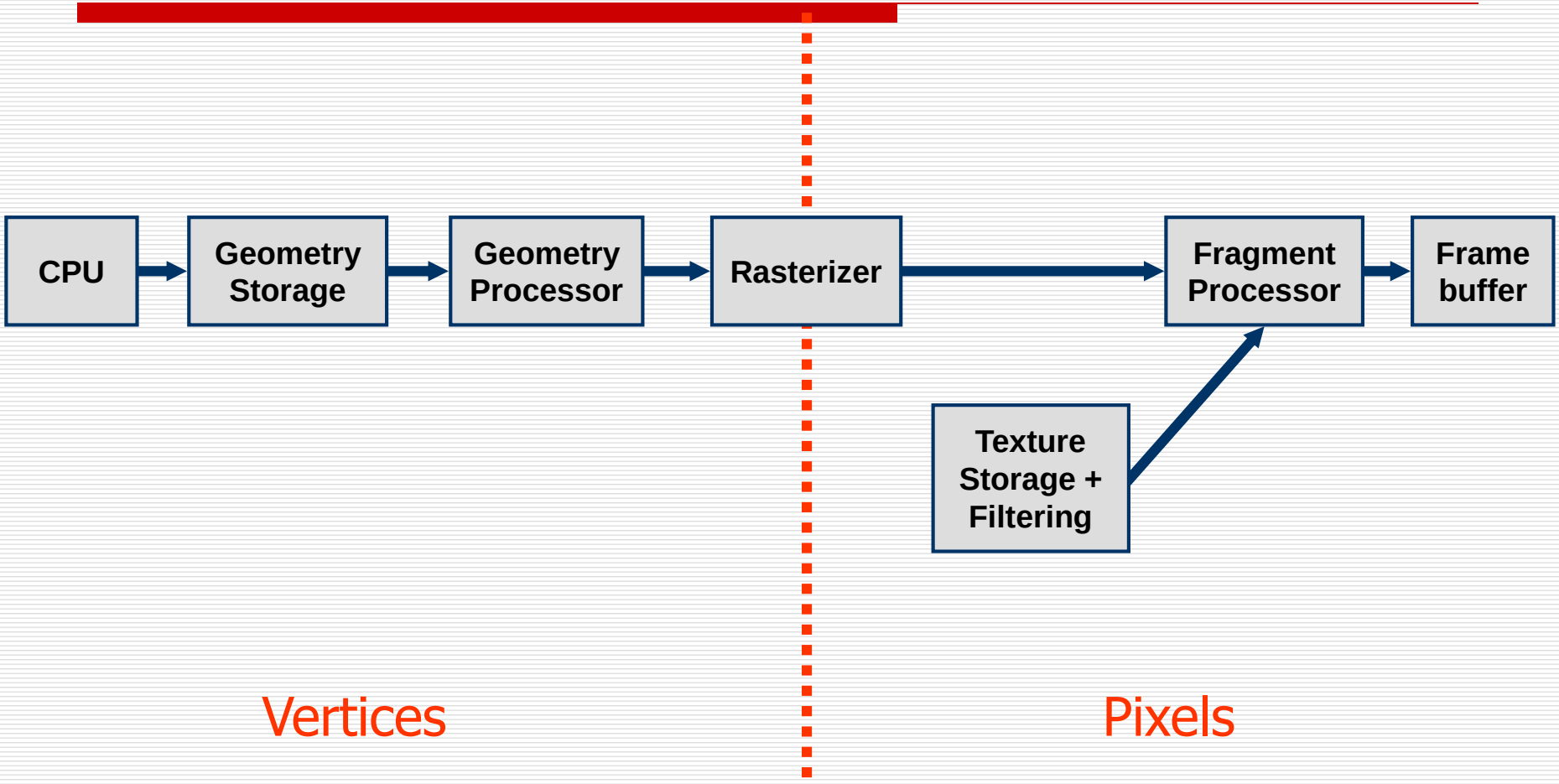
三、高级:GPU硬件加速的流水线



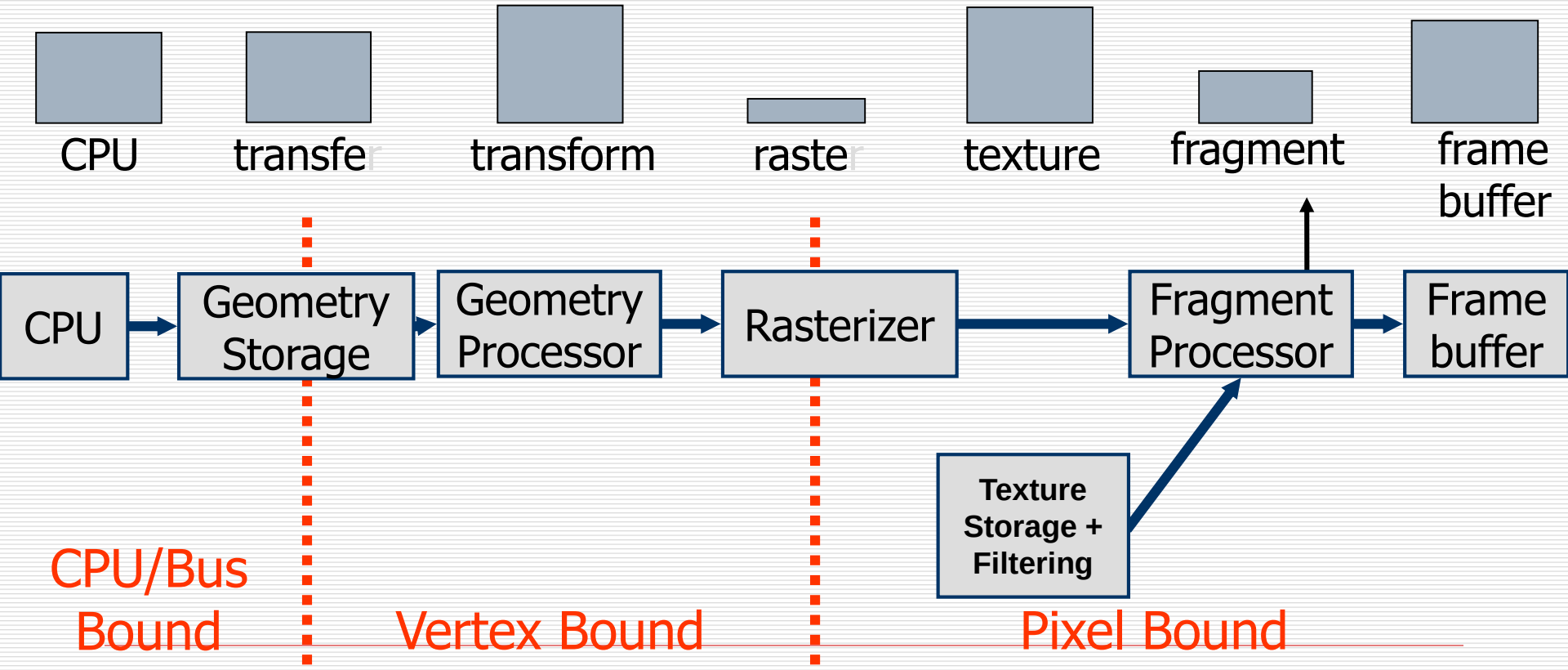
硬件加速

- 硬件加速从流水线的末端开始，然后逐步往高层发展，目前有些应用层的工作已经可以用可编程硬件来实现。

Graphics Pipeline: GPU



Possible Bottlenecks



优点:

- 避免CPU与GPU的交互
- 充分利用GPU的并行性
- 可以提高绘制速度大约10倍.