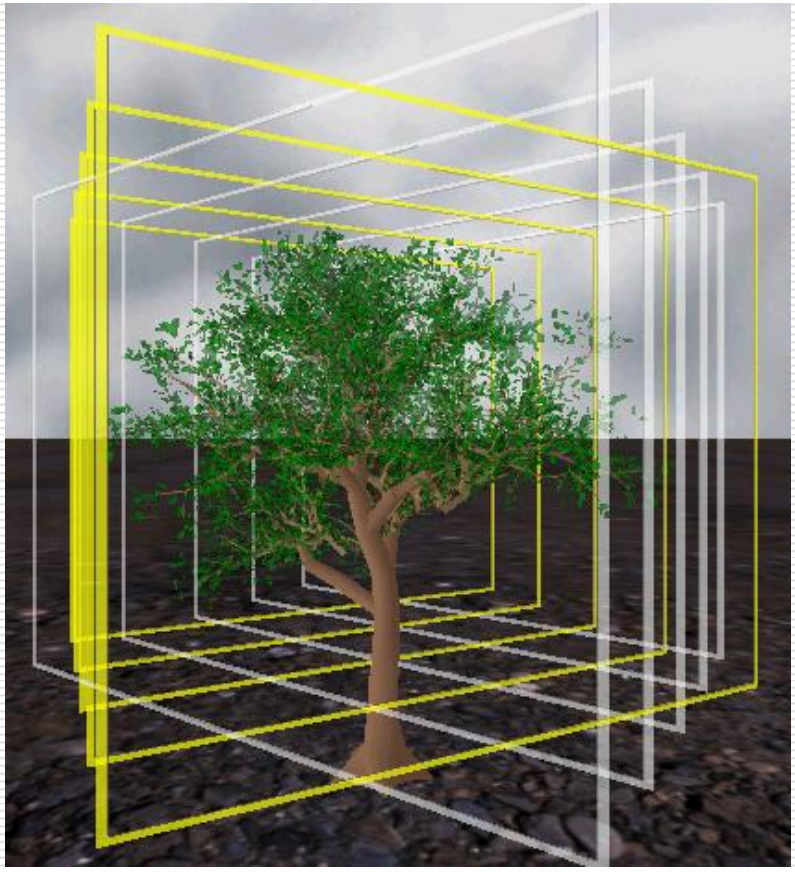
第十一章
动画游戏的特效绘制技术

王长波

一、Billboard技术

Billboard技术

- 几何与图像混合绘制方法；
- 一个带有纹理的四边形，可随着相机的运动而运动，常常与Alpha配合使用；
- 利用很少的几何实现较复杂的效果.



Billboard Configurations

- The billboard polygons can be laid out in different ways
  - Single rectangle
  - Two rectangles at right angles
  - Several rectangles about a common axis
  - Several rectangles stacked
- Issues are:
  - What sorts of billboards are good for what sorts of objects?
  - How is the billboard oriented with respect to the viewer?

Single Polygon Billboards

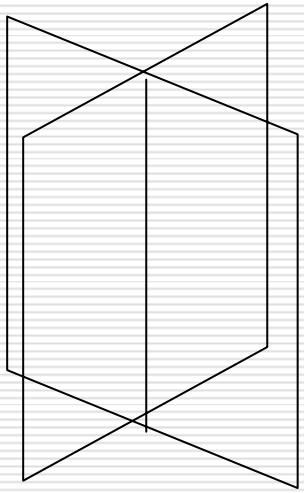
- The billboard consists of a single textured polygon
  - Billboards that are walls, but then they are textured walls!
- Two primary ways of aligning the billboard:
  - Assign an up direction for the billboard, and always align it to face the viewer with up up
  - Assign an axis for the billboard and rotate it about the axis to face the viewer

Aligning a Billboard

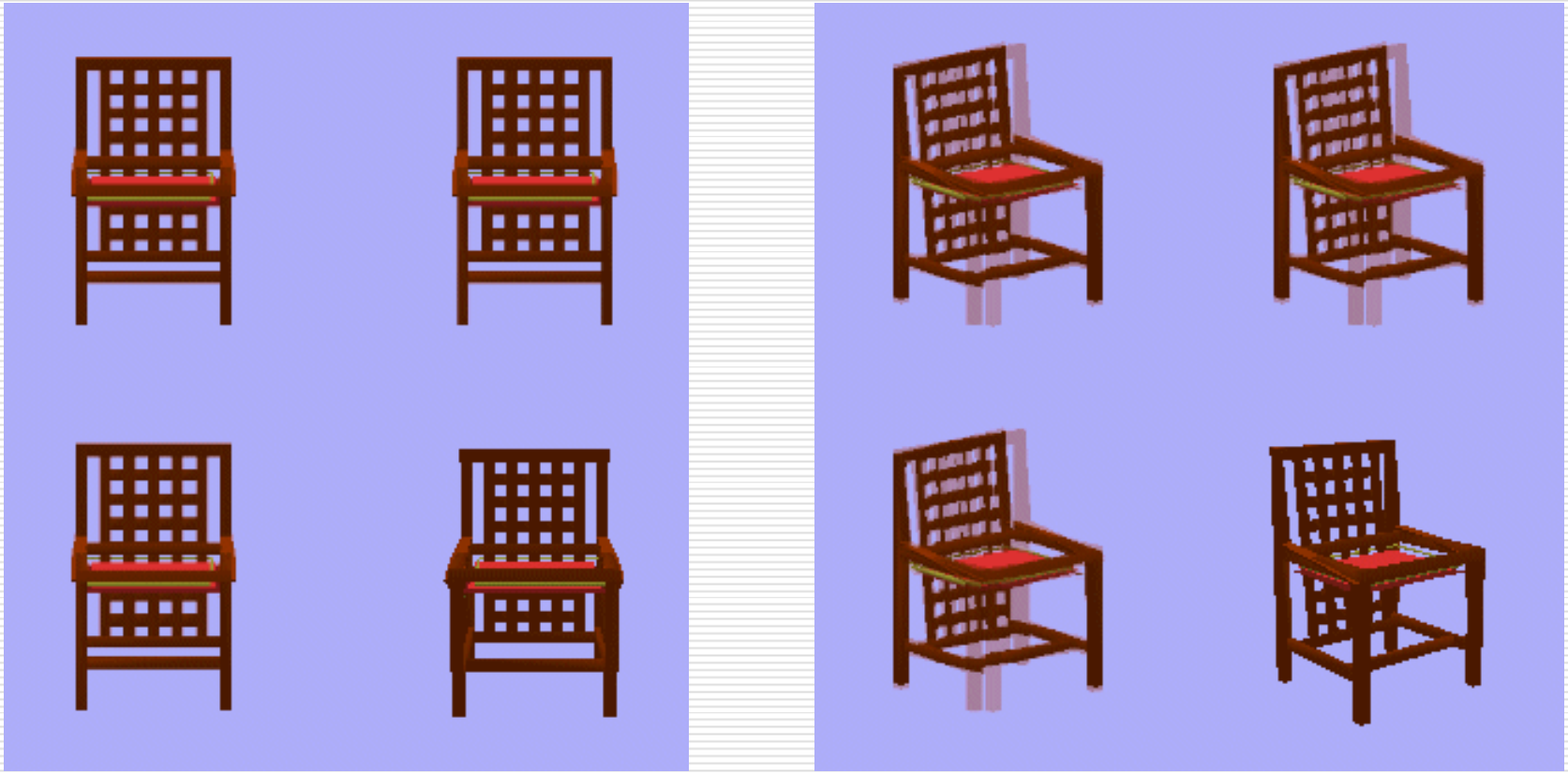
- Facing the viewer and pointing up:
  - 四种：
    - 平行屏幕的Billboard技术：法向与视线重合
    - 平行屏幕的Billboard技术：法向与视线平行
    - 视点朝向的Billboard技术：法向取视点到Billboard中心的连线
    - 轴向Billboard技术
- Rotation about an axis:
  - 利用坐标变换来计算旋转角度

Multi-Polygon Billboards

- Use two polygons at right angles:
  - No alignment with viewer
  - What is this good for?
  - How does the apparent width change with viewing angle?
- Use more polygons is desired for better appearance
  - How does it affect the apparent width?
- Rendering options: Blended or just depth buffered



Sample



Screen shots from an Nvidia demo

Impostor技术

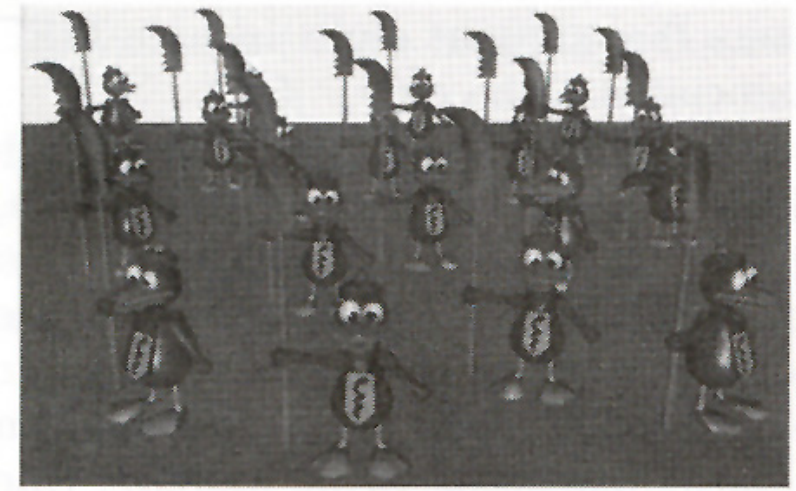
- 替身图方法利用游戏每帧间的连续性，采用二维图像和三维模型的投影替代三维物体；
- 创建替身图：
  - 将三维物体绘制到一个纹理中
- 绘制替身图
  - 相机变焦或物体距离变化时，二维投影尺寸变化；
  - 尺寸大于某一阈值，更换替身图

- Billboard＋精灵动画
  - Perlin噪音函数生成火焰
  - 数个Billboard模拟火苗运动
  - 扰动纹理来实现火苗的串升和风吹效果
  - 模拟用的动态纹理可以采用实拍的视频

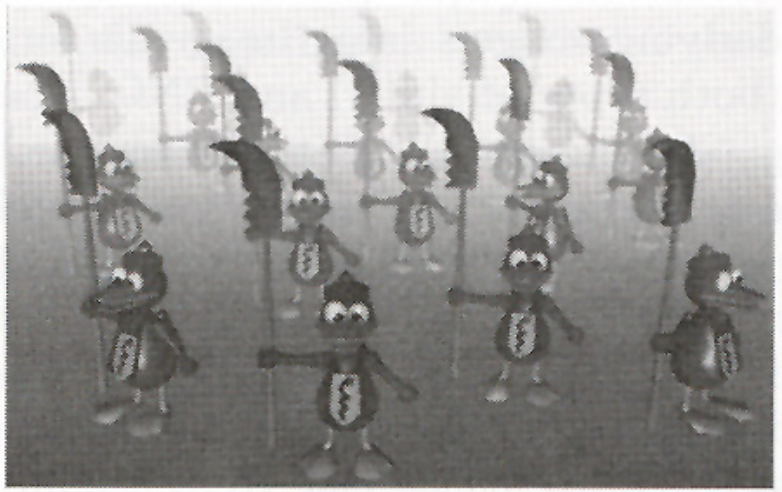
二、雾化绘制

Fog（雾）

- 在图形学中，雾是一种可加到最终图像的大气现象。



无雾效果



有雾效果

雾效果的计算方程

- 假设雾的颜色（由用户指定）为$\mathbf{c}_f$，雾因子（**fog factor**）为f，待绘制物体的颜色为$\mathbf{c}_s$，则象素的最终颜色$\mathbf{c}_p$为：

$$\mathbf{c}_p = f\mathbf{c}_s + (1-f)\mathbf{c}_f$$

- 在上述方程中，f 的值不是很直观，它随离视点的距离而递减。这是**OpenGL**和**DirectX**采用的方程，其好处是可使计算f的方程变得简单。
- 另一种描述方式为$f' = 1 - f$

- 指数雾(**Exponential Fog**):

$$f = e^{-d_f z_p}$$

- 平方指数雾(**Squared Exponential Fog**):

$$f = e^{-(d_f z_p)^2}$$

其中d_f为控制雾浓度的参数。计算得到 f 以后，把结果截取(**clamp**)到[**0,1**]

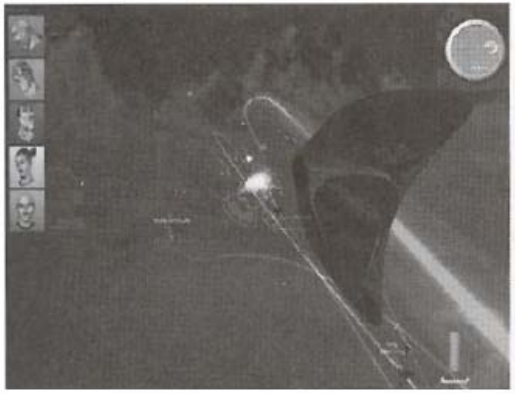
如何加雾：

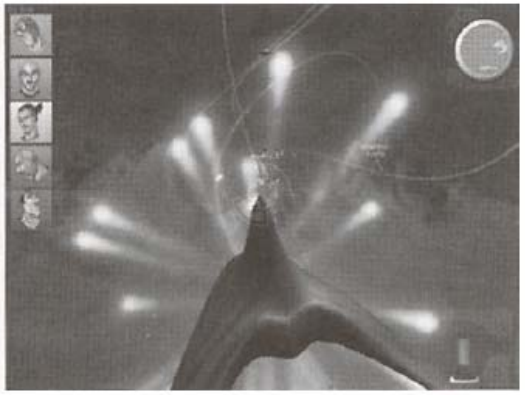
```
    //设置雾的状态打开
SetRenderState(D3DRS_FOGENABLE,TRUE);

    //设置雾的颜色，起始点，密度信息
SetRenderState(D3DRS_FOGCOLOR,RGB(225,225,225));
SetRenderState(D3DRS_FOGSTART,*(DWORD *)(&Start));
SetRenderState(D3DRS_FOGEND,*(DWORD *)(&End));
SetRenderState(D3DRS_FOGDENSITY,*(DWORD *)(&Density));

    //设置雾的变化方式
if( mode == D3DFOG_LINEAR)
    {
        SetRenderState(D3DRS_FOGVERTEXMODE,mode);
    }
else if( mode == D3DFOG_EXP)
    {
        SetRenderState(D3DRS_FOGVERTEXMODE,mode);
    }
```

三、动态特效绘制

Billboard技术

- 几何与图像混合绘制方法；
- 一个带有纹理的四边形，可随着相机的运动而运动，常常与Alpha配合使用；





Impostor技术

- 替身图方法利用游戏每帧间的连续性，采用二维图像和三维模型的投影替代三维物体；
- 创建替身图：
  - 将三维物体绘制到一个纹理中
- 绘制替身图
  - 相机变焦或物体距离变化时，二维投影尺寸变化；
  - 尺寸大于某一阈值，更换替身图

- Imposter是从当前视点出发，将一个复杂的物体绘制到一张图像上，作为texture贴到一个多边形上，如billboard
- 动机：显示一张图像的时间远小于画一大堆多边形的时间。其代价为需要更新的图像

优点：

绘制远处的物体很有效

一堆小的静态物体

不以牺牲细节为代价

- 更新是绘制中最为重要的计算
- 游戏
- 启发式修改纹理坐标已减少视差错误
  - 预定义一个最大距离，当经过n帧以后，其在屏幕上的阈值，则更新
  - 为了实其更有效，每次绘制一个大的纹理中的一些子区域
- 动态物体
  - 当其运动时，使用3D模型
  - 当其静止时，使用imposter

粒子系统

- 粒子系统是一系列独立个体的集合,它们以一定的物理规律和生命周期在场景中运动.
- 粒子具有一些属性：位置，速度，加速度，能量，方向等
- 在粒子运动过程中，粒子属性被显示，修改和更新
- 粒子的变化规律受到物理规律的影响
- 既有随机性，又有规律性
- 可以用来模拟烟，火焰，爆炸，雨雪，血溅等

- 基本过程
  - 初始化粒子
  - 当程序运行时
    - 如果粒子没有消亡
      - 根据粒子的速度更新粒子的位置
      - 根据粒子的加速度更新粒子的速度
      - 修改粒子的能量
    - 如果粒子的能量小于某一阈值
      - 设置粒子的状态为消亡
    - 如果粒子击中场景物体或其他粒子
      - 修改粒子的位置，方向，速度和能量
    - 显示粒子
  - 程序结束

难点

- 粒子：长方形，Pointsprite和三维几何模型，Billboard
- 初始化函数和状态更新函数

- 加快速度：粒子消亡后设置初始化，不释放出内存
- 与游戏引擎的无缝连接

爆炸

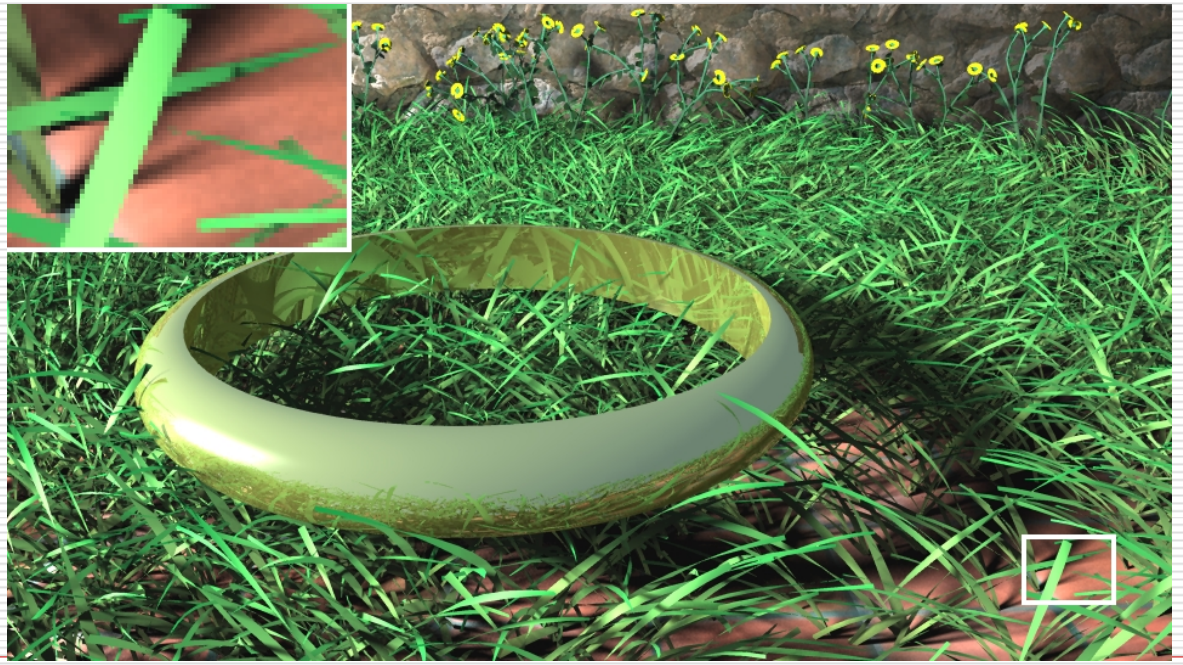
- 最常见的就是采用粒子系统模拟，也有采用Billboard模拟爆炸的。
- 粒子分步在以爆炸点为中心的球面，从球心以高速和巨大的能量向外发射
- 有遮挡的爆炸：在半球面上分布粒子

火焰

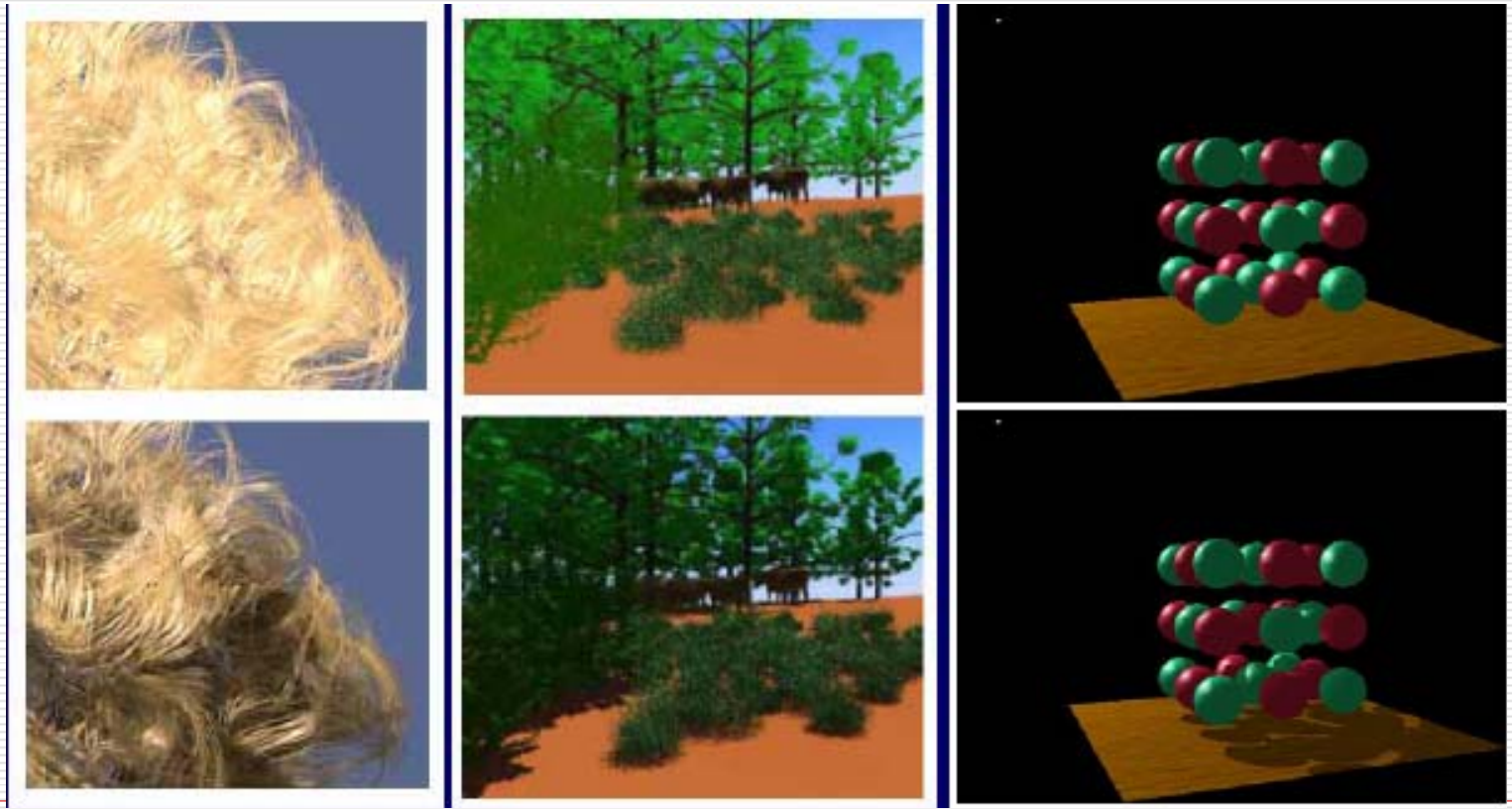
- 二维随机函数
  - 设置火源：在预期的位置生成火焰的中心
  - 选择随机函数生成火焰的形状和颜色
  - 选择正确的图像模糊算法，模拟火焰热源扩散

- Billboard＋精灵动画
  - Perlin噪音函数生成火焰
  - 数个Billboard模拟火苗运动
  - 扰动纹理来实现火苗的串升和风吹效果
  - 模拟用的动态纹理可以采用实拍的视频

四、阴影绘制算法

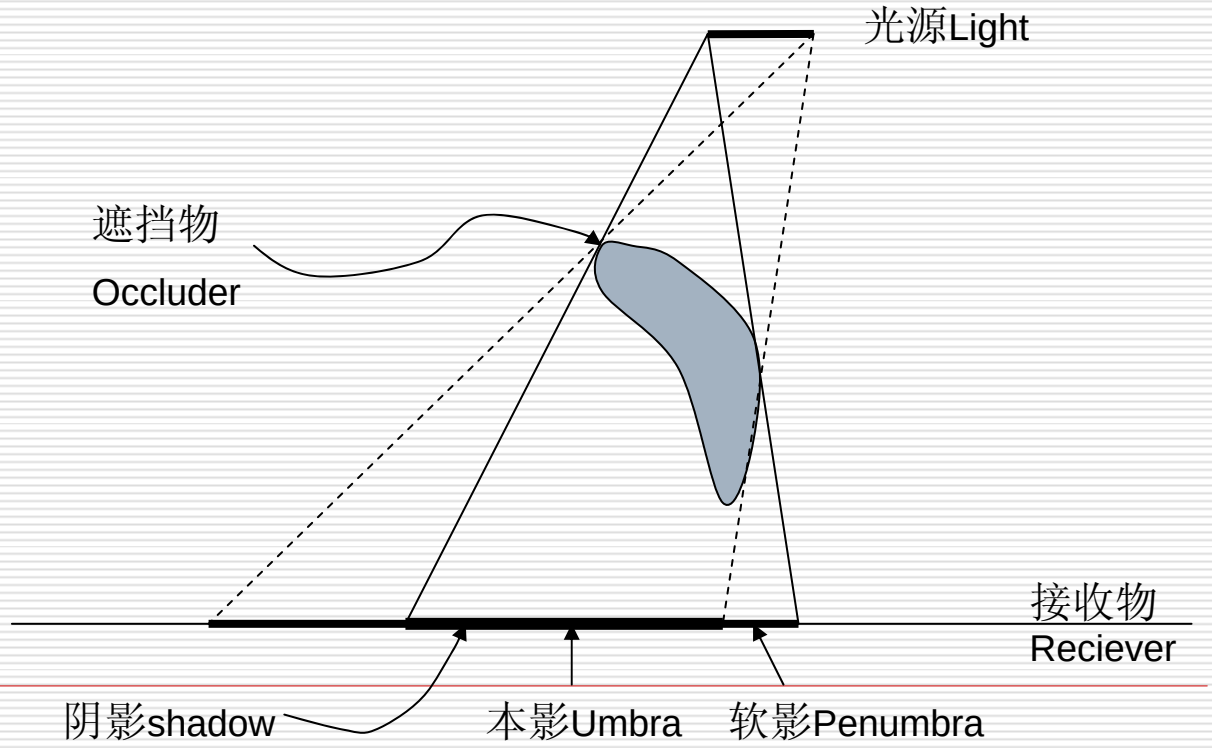


- 阴影使效果更逼真



阴影术语

- 阴影效果在真实感图像的生成和物体位置的判断方面是一个非常重要的元素
- 有关阴影的一些术语如图



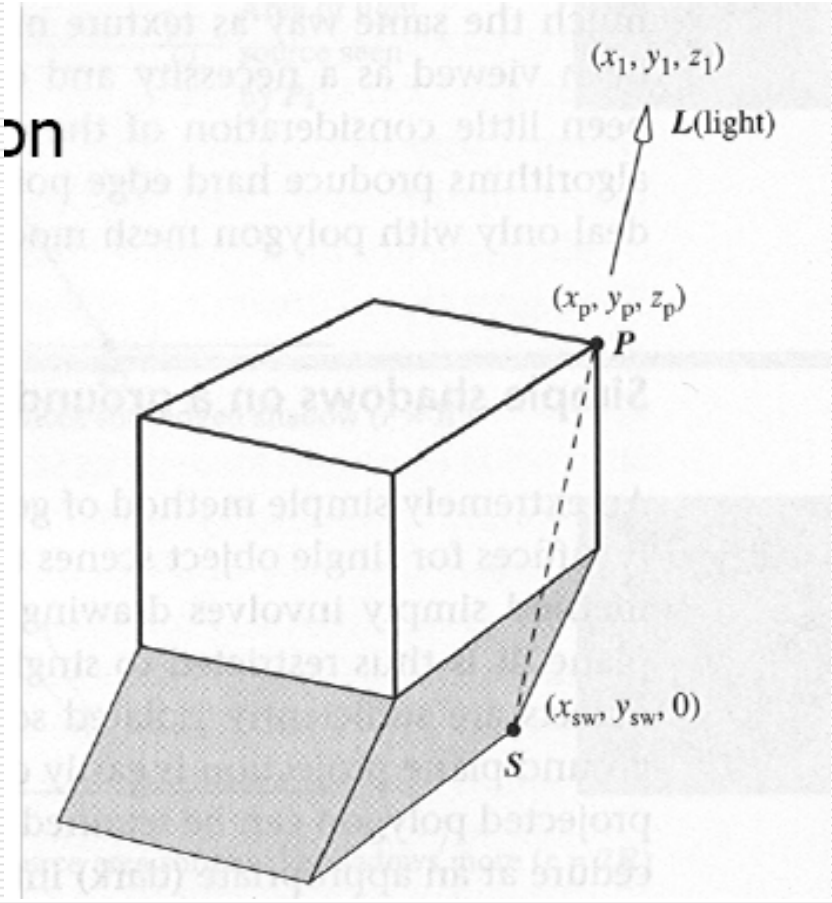
- Hard shadow（本影）
  - for point light source or light source at very
  - large distance
  - Can sometimes be misinterpreted as geometric features.
- Soft shadow （软影）
  - For area light source
  - Generally preferable

经典的三种本影阴影方法

- 投影阴影
- 阴影图的方法
- 阴影体的方法

投影阴影算法

在这个方法里通过对一个三维的物体的两次绘制来产生一个阴影。第一次绘制画出物体。第二次将物体上的点通过投影矩阵绘制在要投影的平面上就生成阴影了。



- 关键就在计算出一个投影矩阵。投影矩阵的一般形式：假设要投影的平面 π 的方程是：**n*x+d=0** 其中**n**是平面法向量，光源在**l**处，物体上点**v**在平面 π 上的投影坐标点为**p,**则满足**M*v = p**其中**M**就是投影矩阵。要绘制阴影只要简单的把这个矩阵应用到要在平面 π 上投影的物体上，再用黑色对物体进行绘制就可以了。

$$\mathbf{M} = \begin{bmatrix} \mathbf{n}*\mathbf{l}+d-l_x n_x & -l_x n_y & -l_x n_z & -l_x d \\ -l_y n_x & \mathbf{n}*\mathbf{l}+d-l_y n_y & -l_y n_z & -l_y d \\ -l_z n_x & -l_z n_y & \mathbf{n}*\mathbf{l}+d-l_z n_z & -l_z d \\ -n_x & -n_y & -n_z & \mathbf{n}*\mathbf{l} \end{bmatrix}$$

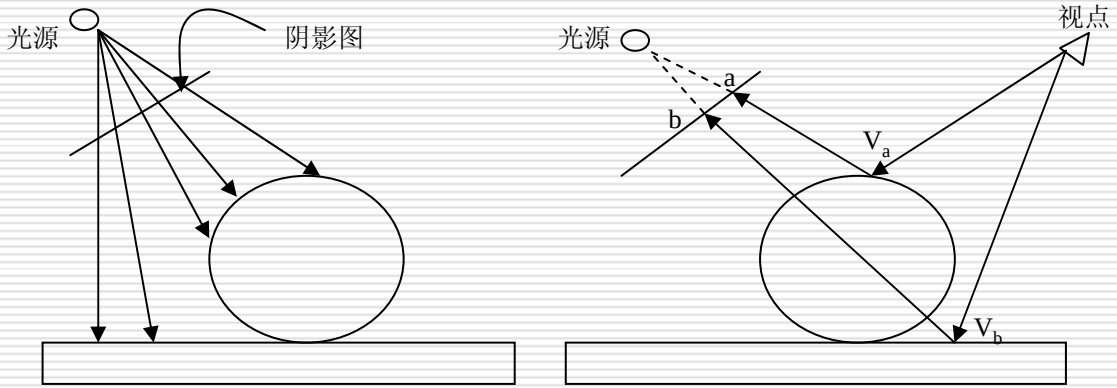
阴影图（shadow map）算法

- 这种方法的主要思想是使用Z缓冲器算法，从投射阴影的光源位置处对整个场景进行绘制。对于Z缓冲器内的每一个像素，它的Z深度值包括了这个像素到距离光源最近的物体的距离。一般将Z缓冲器中的整个内容称为阴影图（shadow map）,有时候也称为阴影深度图或者阴影缓冲器。

- 为了使用阴影图，需要将场景进行第二次绘制，不过这次是从视点的角度来进行的。现在，在对每个图元进行绘制的时候，将他们的位置与阴影图进行比较，如果绘制点距离光源比阴影图中的数值还要远，那么这个点就在阴影中，否则就不在阴影中。具体见下图。

阴影图（shadow map）算法

- 图中，左边表示将点的深度值保存在可见的物体表面上，形成一个阴影图。右图中眼睛正在观察的两个位置，点Va，Vb，分别对应阴影图上的点a，b。Va处到光源的距离不大于阴影图a中的保存的数值，所以Va被照亮。点Vb到光源的距离大于阴影图中b点保存的深度值，所以点Vb在阴影中。

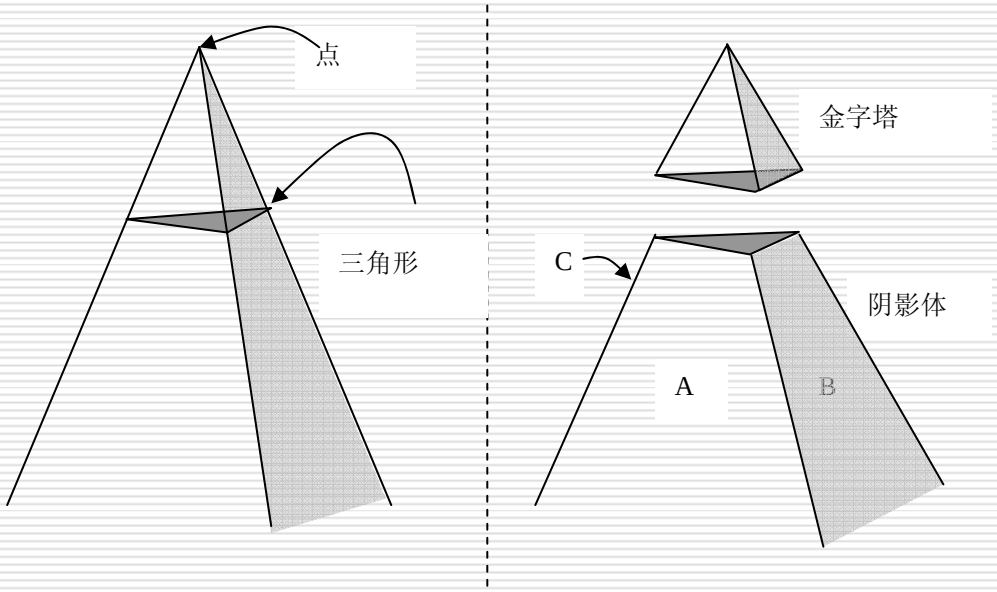


阴影体（shadow volume）算法

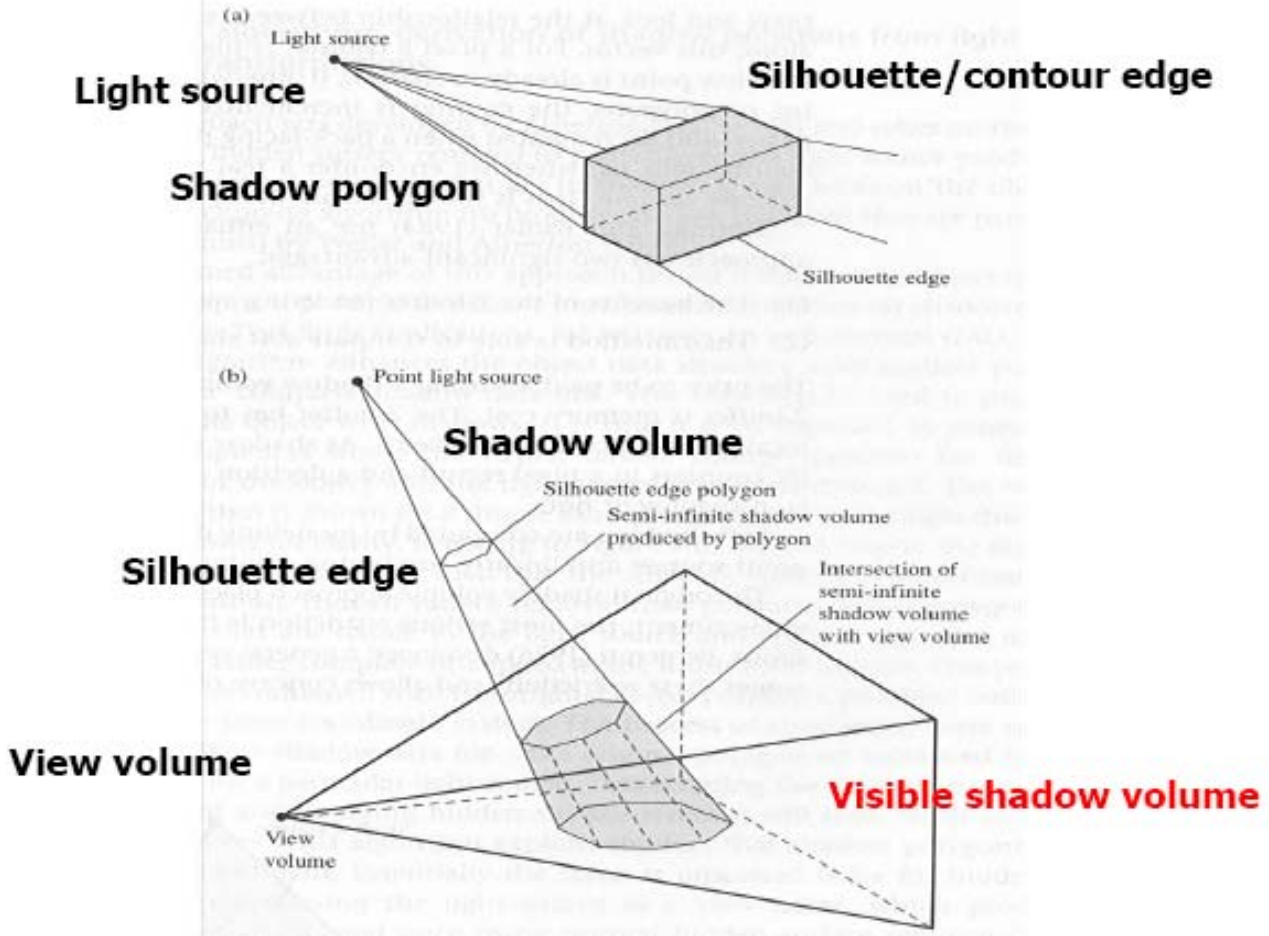
- 在1991年，Heidmann提出了一种基于Crow阴影体的方法，可以将阴影投影到任意形状的物体上，我们称之为阴影体方法。

- 在开始前，我们来了解一下什么是阴影体。假设有一个点和一个三角形，然后从这个点向三角形的3个顶点分别进行连线，这三条连线延伸向无穷远，这样就形成一个无穷体积的金字塔体。在三角形平面以下的部分是一个切去顶端的无穷金字塔体，而三角形上面是一个简单的金字塔体，如图下图。

阴影体（shadow volume）算法



现在，假设这个点是点光源，那么所有在被横切了的金字塔体内（三角平面以下）的物体都处于阴影中，这个体就称为阴影体。


(a)
Light source
Light source
Silhouette/contour edge
Shadow polygon
Silhouette edge
(b)
Point light source
Shadow volume
Silhouette edge polygon
Semi-infinite shadow volume
produced by polygon
Silhouette edge
Intersection of
semi-infinite
shadow volume
with view volume
View volume
Visible shadow volume
View
volume

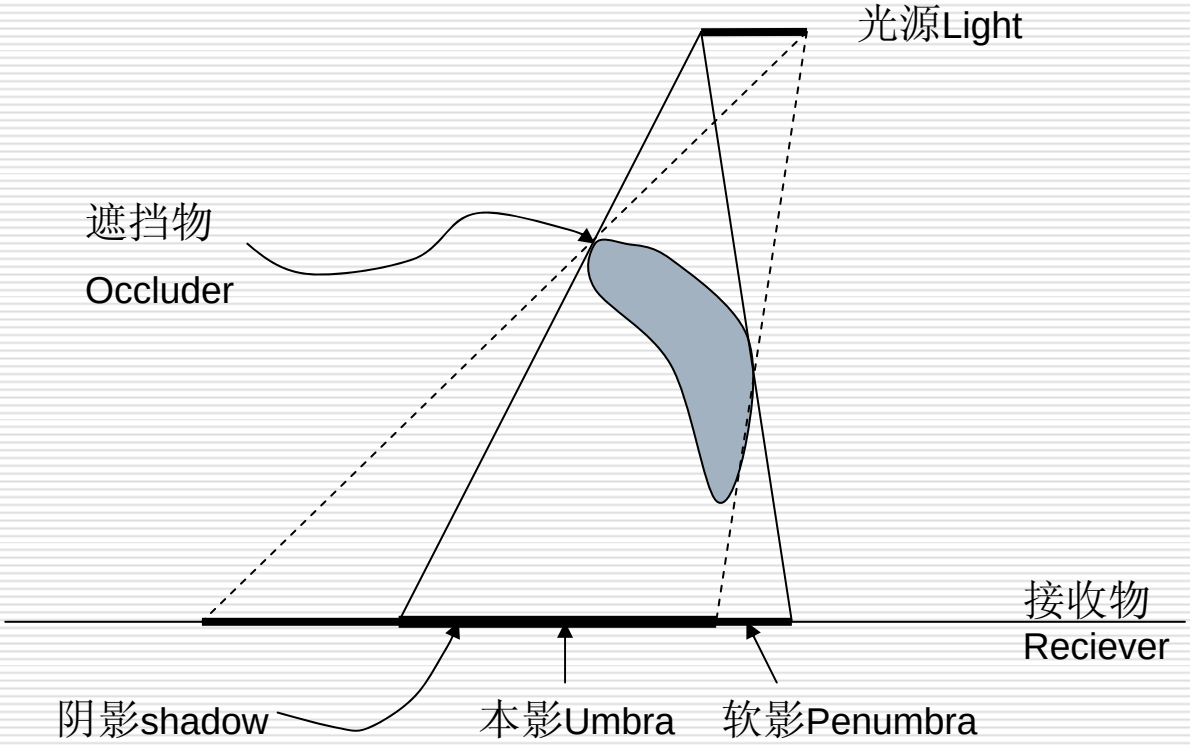
阴影体（shadow volume）算法

- 阴影体本身并不被绘制，只用来决定其他物体是否处于阴影中。
- 相对于观察者，一个正面阴影多边形（上右图中A,B）使得在它之后的物体被投上阴影；一个背面阴影多边形（上右图中C）则忽略了正面的这种效应。
- 考虑从视点V到物体上一点的向量。如果该向量与阴影体相交的正面多于背面，那么该点处于阴影中。
- 我们可以设置一个计数器来跟踪这个向量，如果该向量穿过了一个阴影体正面，那么计数器加1，穿过背面的时候减去1，最后这个向量到达这个像素处要显示的物体。如果最终计数器的数值大于零，那么说明这个像素点处于阴影中，否则就能被光照亮。

比较

- 投影阴影算法的最大局限性在于它只能在平面上生成阴影，无法满足我们在曲面上生成阴影的要求。
- 阴影图算法由于在比较过程中要对阴影图进行采样，因此这种算法容易产生走样问题。例如我们并不能保证光源采样与屏幕采样的点在完全相同的位置上。
- 阴影体算法很精确，但它所需要的开销比较大，处理过程冗长，不适合我们实时生成阴影的要求。

软影的生成


光源Light
遮挡物
Occluder
接收物
Reciever
阴影shadow
本影Umbra
软影Penumbra

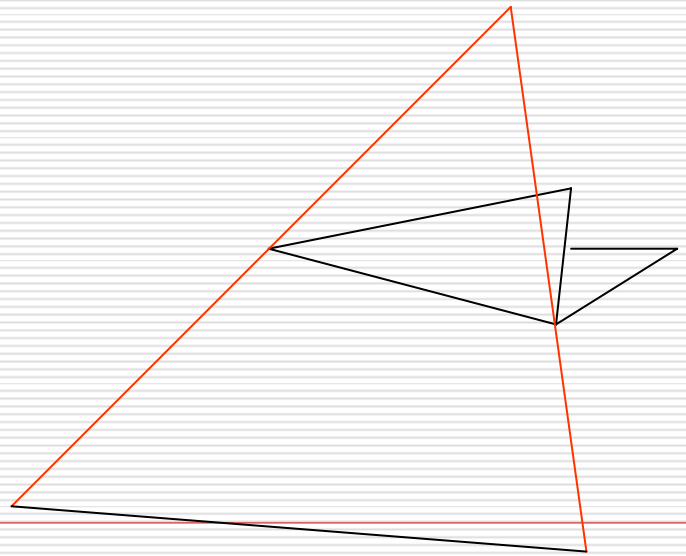
基于轮廓的动态软影生成算法

- 找出物体外轮廓
- 在平面上产生阴影
- 将阴影做为纹理映射到曲面上
- 反走样等后期处理

1.找出物体外轮廓

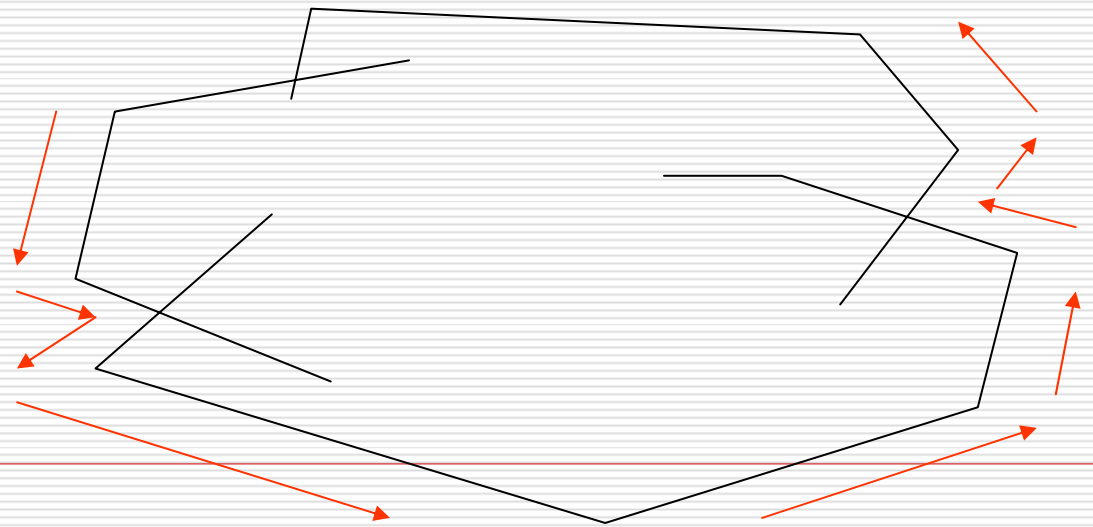
- 找到光源坐标系下的物体外轮廓

  一条边所对应的两个三角形的另外两个点在光源和这条线构成平面的同一侧



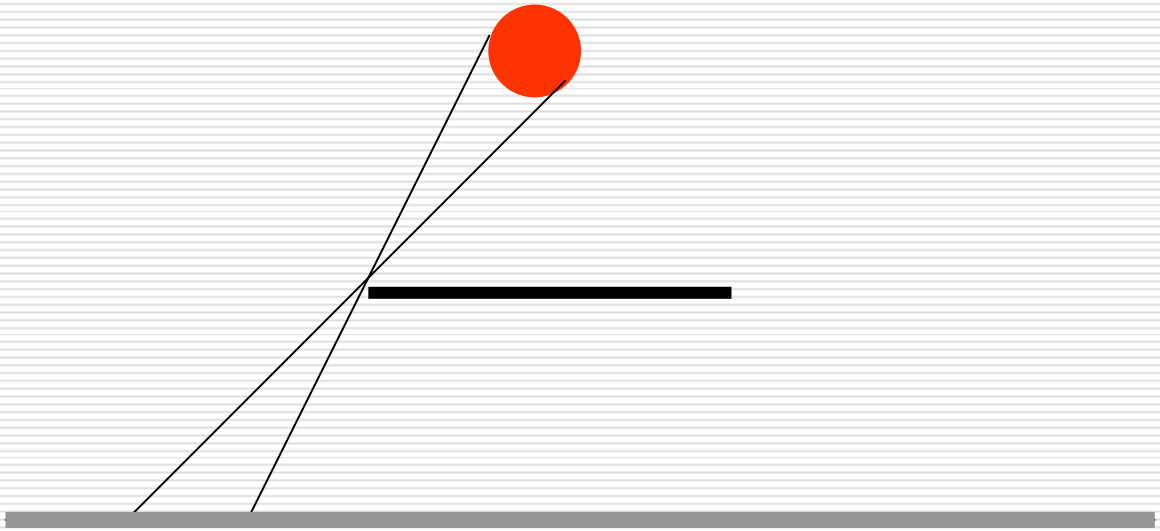
2.在平面上生成阴影(一)

- 将轮廓线投影到投影面上，进行填充。



2. 在平面上生成阴影(二)

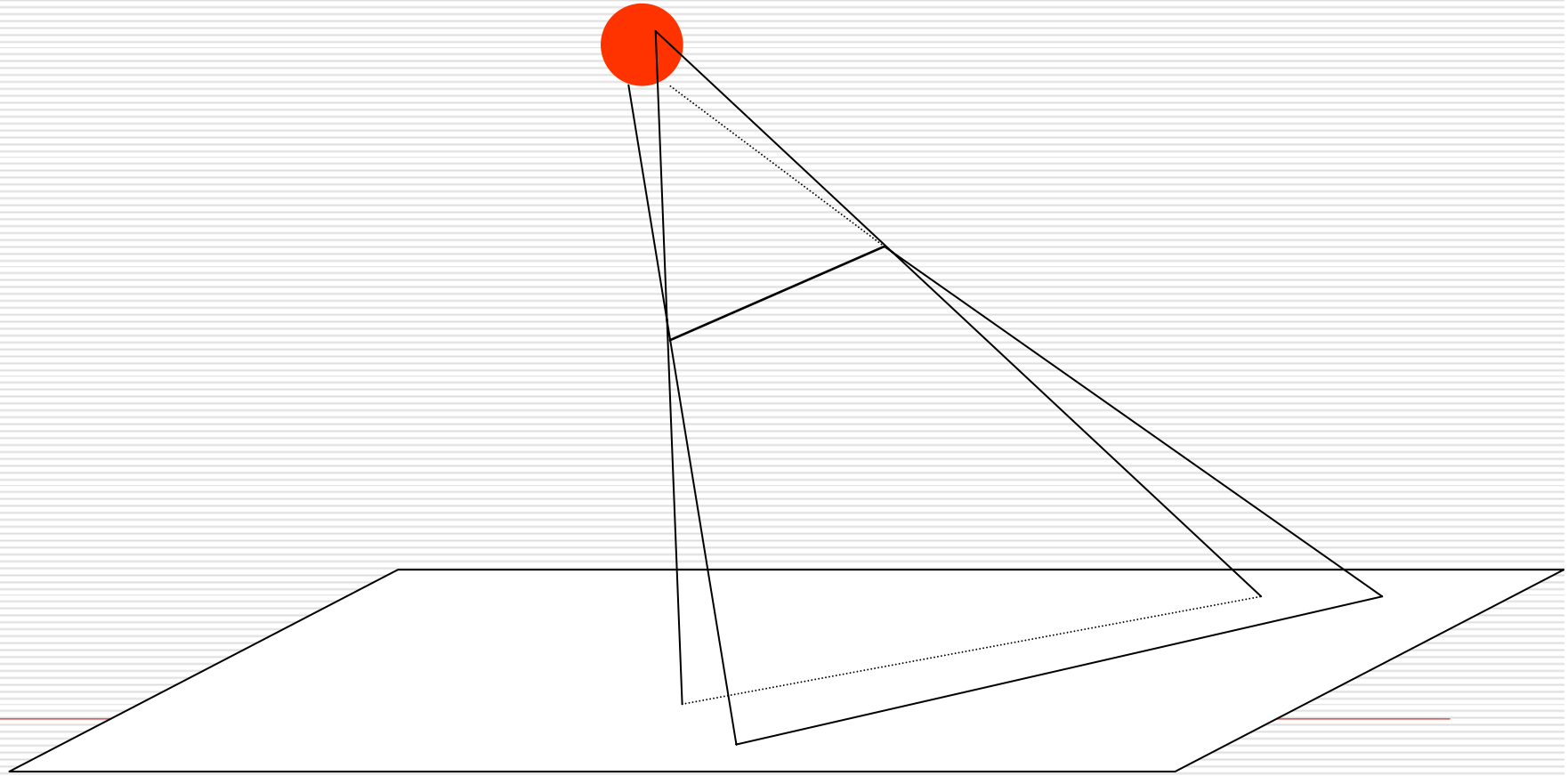
- 上一步中生成的是实阴影(hard shadow)，接下来我们在这个基础上生成半影(也称软影)



2.在平面上生成阴影(二)

- 我们将在空间的一条轮廓线投影到投影面上，这样每一条轮廓线会产生两条投影线，一条是最亮的边界(光线全部都可以到达)，另外一条是最黑的边界(光线都不能到达)。而中间的话，就是产生半影的区域。我们采用特殊的方法对半影区域进行填充，达到精确而又真实的投影图。

生成半影区示例图



3. 在曲面上生成阴影

- 计算投影平面在曲面上的交点，得到曲面上阴影轮廓。先得到平面上阴影轮廓上点的坐标，然后计算通过这个点出发的平面法向量与曲面的交点，由此得到曲面上的阴影轮廓。
- 得到曲面上点与纹理的相应坐标后就可以在曲面上贴阴影纹理。

4.后期处理

- 反走样处理

  曲面上阴影的边缘处对软影的处理可能会比较麻烦，有可能一个曲面的三角型面片被分割成多个部分，对这个三角型的贴图就要分步处理。

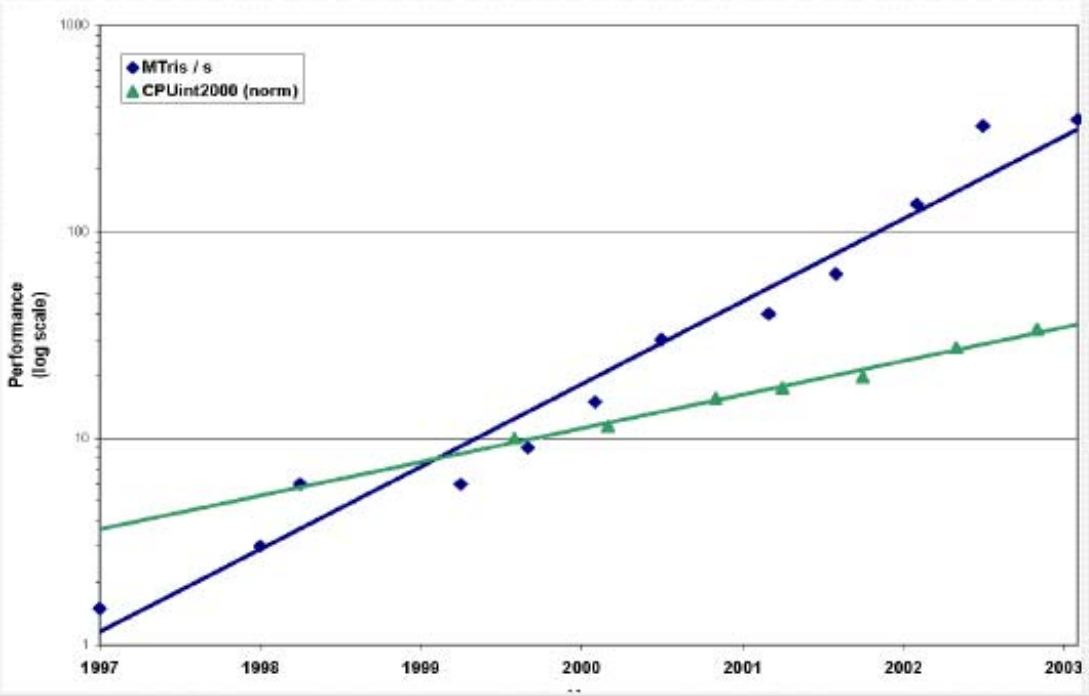
- 帧与帧之间的连贯性

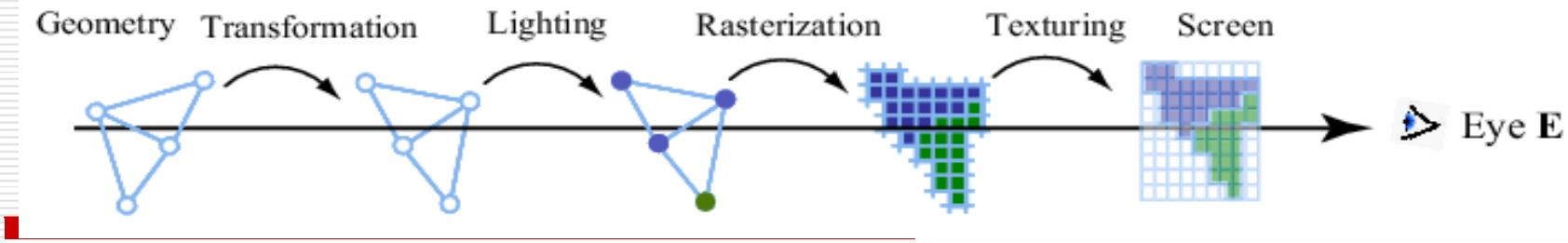
  可以利用帧与帧之间的连贯性对计算进行加速，比如对5帧以内的图像取平均来加速。

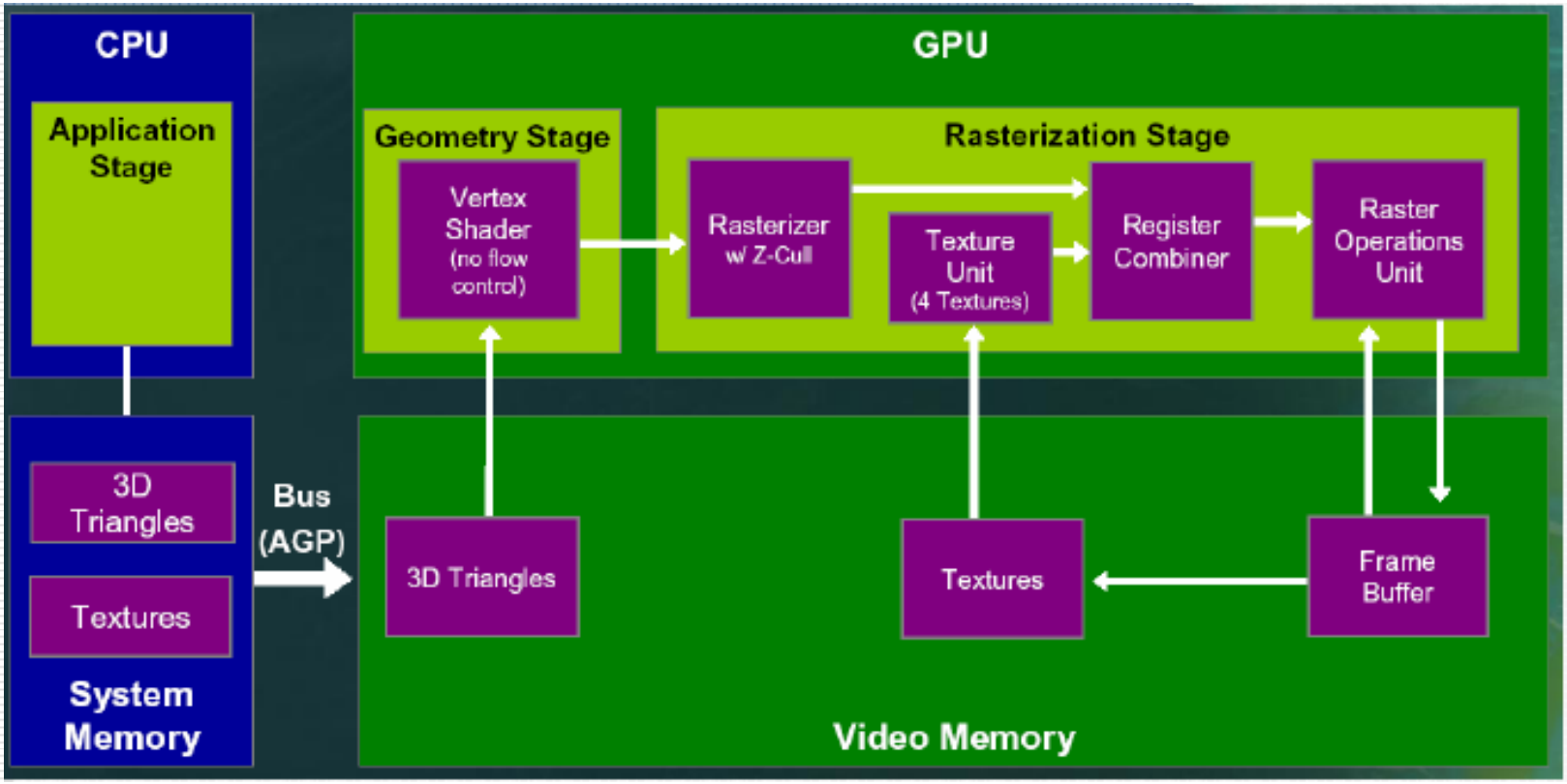
五、GPU绘制技术

计算机图形处理器（GPU）近年的发展

-- GPU的发展速度远远超过CPU发展的摩尔定律

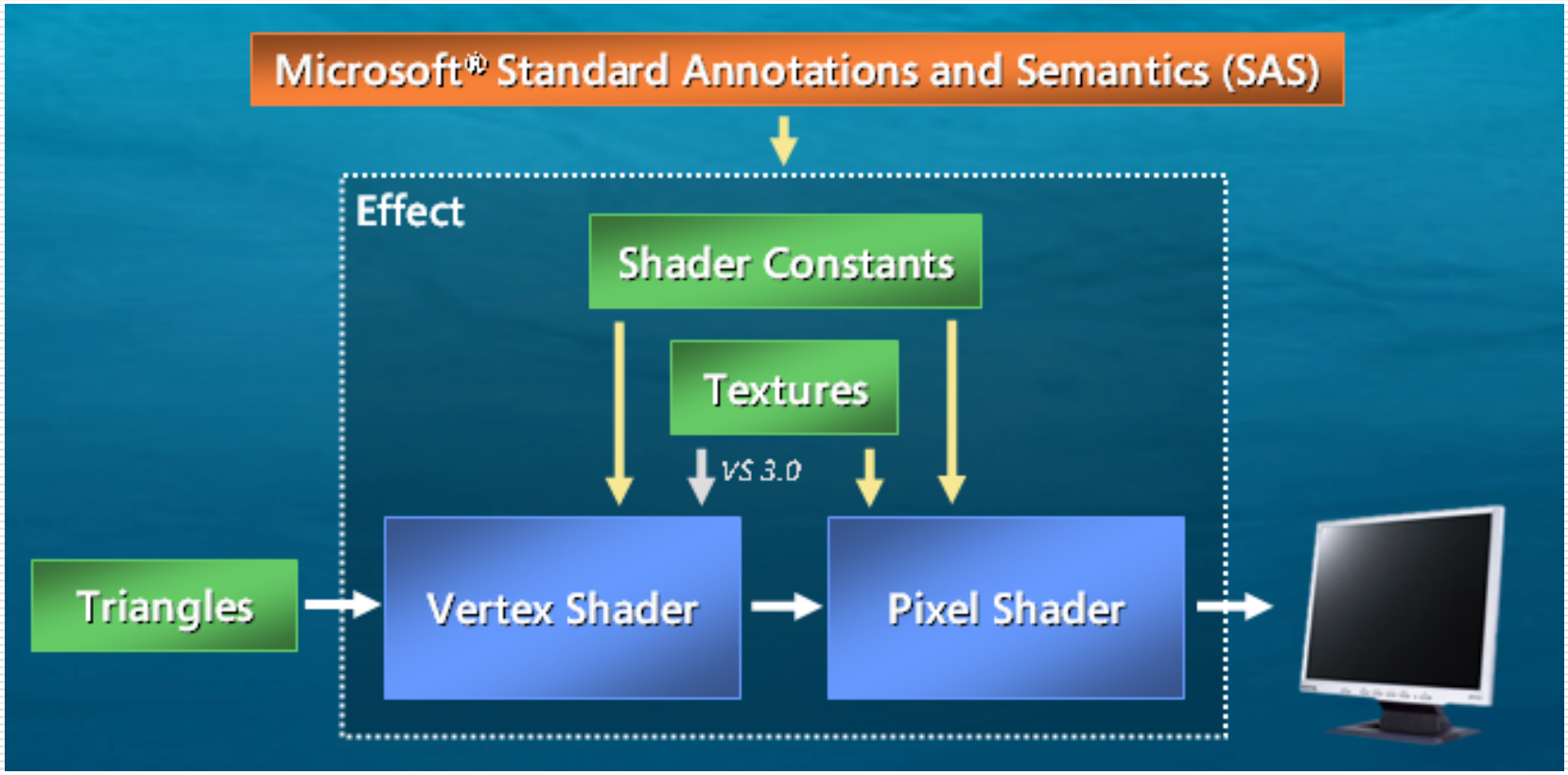






最大瓶茎: CPU与GPU之间进行数据交换

The Programmable Pipeline

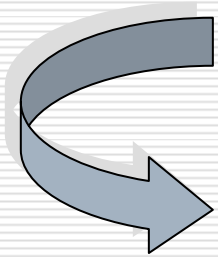


GPU 的优点:

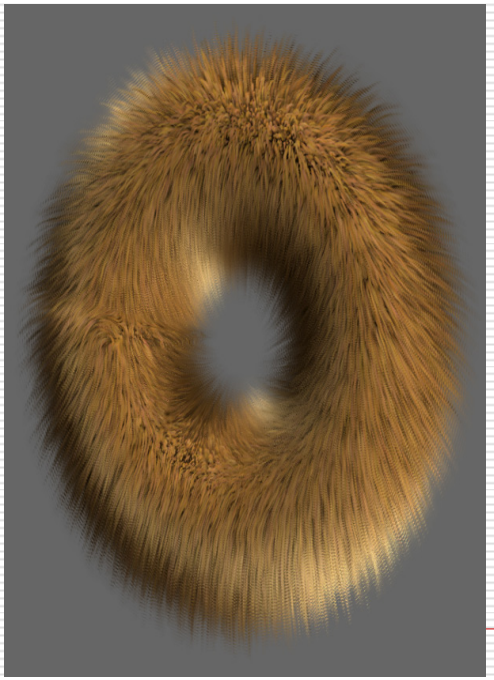
- 不需要在**CPU**与**GPU**之间进行数据交换
  - 可以提高速度十倍以上.
- **SIMD**的并行处理, 多管道流处理器
  - 图形卡内部的内存位宽达**256**位; 纹理**(512M)** 作为内存来使用**--** 高密集的运算.
- 在顶点级和像素级提供了灵活的可编程特性, 可以生成一些特效.

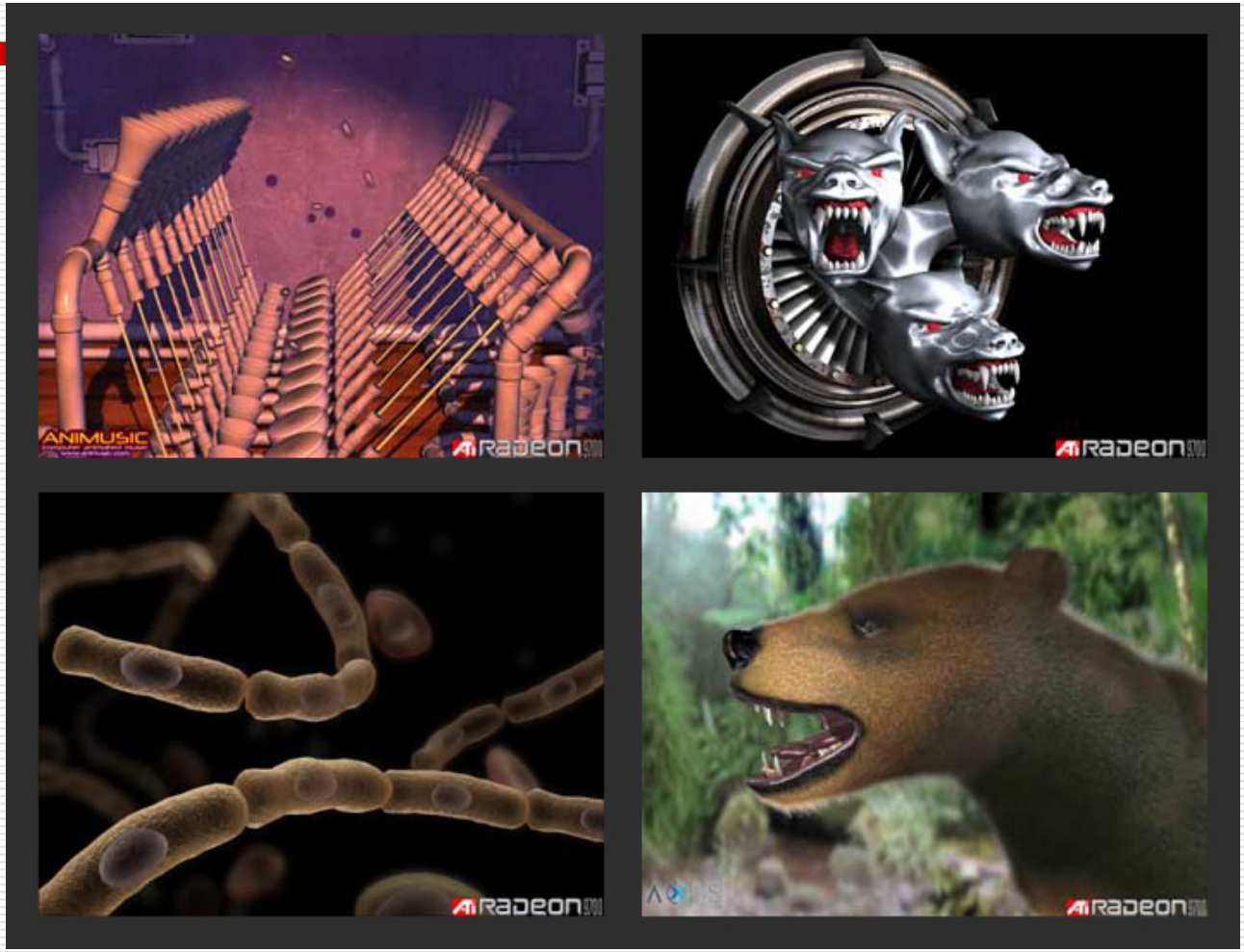
GPU的用途:

- 高效的可编程流线型处理器（无指针）
- 向量形式处理（位置、颜色、纹理坐标)
- 强大的浮点处理能力
- 高内存带宽(>27.1GB/s)
- 硬件增长比摩尔定律快



- 影院级真实效果绘制
- 实时绘制(15~60fps)
- 基于物理原理的复杂自然现象模拟




ANIMUSIC
ATI RADEON 9700
ATI RADEON 9700
ATI RADEON 9700
ATI RADEON 9700

GPU组成:

顶点处理器 (Vertex Shader)

- 多管道(4-6)并行处理(每个管道有 RGBA四个颜色通道同时计算)
- 支持浮点运算, 动态和静态控制流的循环和分支操作，以及子函数操作
- 顶点纹理……

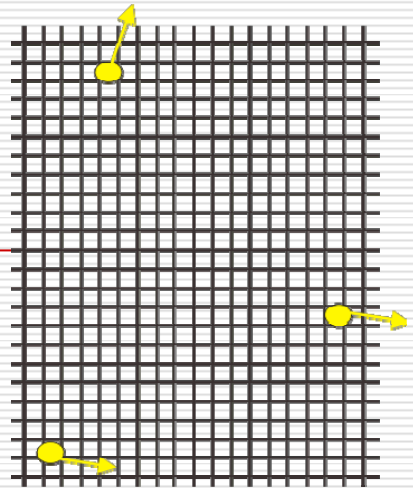
像素处理器 (Pixel/Fragment Shader)

- 多管道(4-16)并行处理(每个管道有 RGBA四个颜色通道同时计算)
- 支持IEEE32 浮点运算, 循环和分支操作
- 将纹理作为内存来使用,支持绘制到纹理的功能及纹理操作
- Multiple Render Target……

NVIDIA

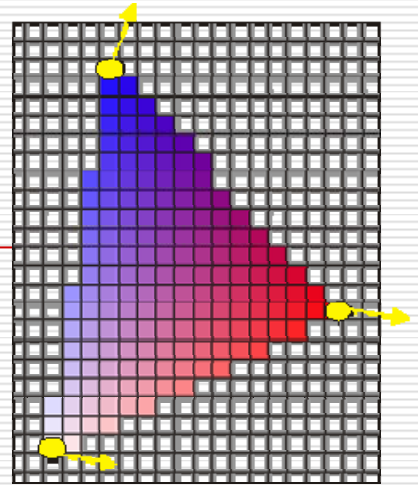
- SLI（Scalable Link Interface）
- CineFX3.0
- SM3.0

Vertex Shader



- Input:
  - Vertex stream
  - Shader constants
  - Texture data (VS 3.0)
- Output:
  - *Required:* Transformed clip-space position
  - *Optional:* Colors, texture coordinates, normals (data you want passed on to the pixel shader)
- Restrictions:
  - Can’t create new vertices
  - Can’t store result in memory

Pixel Shader



- Input:
  - Interpolated data from vertex shader
  - Shader constants
  - Texture data
- Output:
  - *Required:* Pixel color (with alpha)
  - *Optional:* Can write additional colors to multiple render targets
- Restrictions:
  - Can’t read and write the same texture simultaneously

- 编程语言：
  - CG：C语言语法
  - Shader language：for OpenGL
  - HLSL：D3D
  - FX：D3D

顶点渲染

- 查询顶点渲染版本
- 创建顶点缓冲区和顶点声明对象
- 创建顶点渲染对象
- 设置顶点渲染函数并绘制图元

顶点渲染指令简介

- 顶点渲染版本声明
- 顶点元素声明
- 渲染处理

像素渲染

像素渲染过程

- 查询像素渲染版本
- 创建像素渲染指令
- 编译并载入像素渲染指令
- 创建像素渲染对象
- 执行像素渲染

像素渲染指令简介

- 版本声明指令
- 常量定义和寄存器声明指令
- 像素渲染处理指令
- 像素输出指令

像素渲染指令控制

- 指令修改符
- 源寄存器修改符
- 目标寄存器元素屏蔽
- 源寄存器元素替换

一个D3D漫游系统的绘制

- 建模:
  - 模型导入
  - 程序生成
- 绘制:
  - 纹理映射
  - 特殊技巧
- 漫游控制:
  - 第一人称
  - 第三人称视角

END