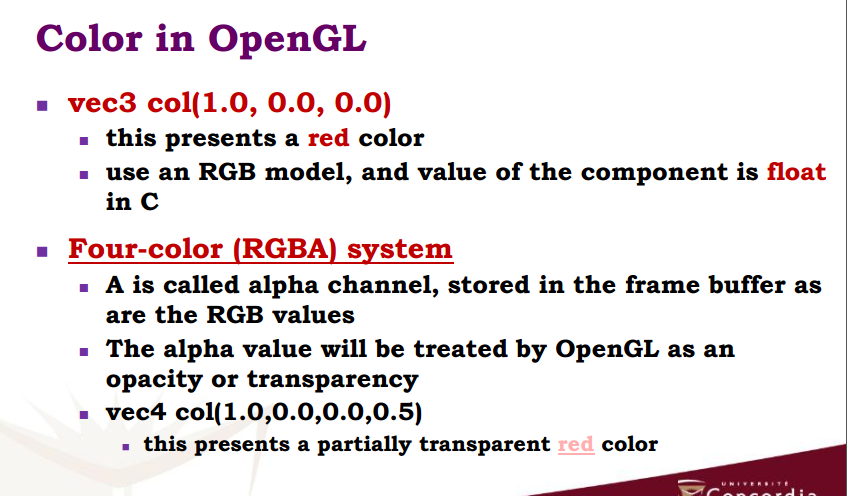
OPENGL里的颜色



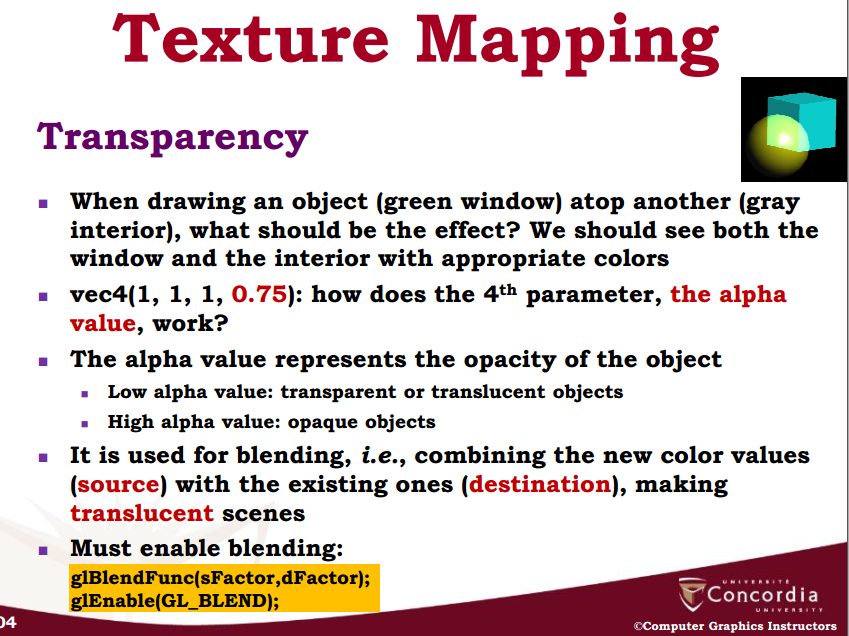
第一个代表红色，float

第二个是RGBA 系统

A叫做alpha channel，1代表不透明opacity，0代表透明transparency

0.5代表部分透明（粉色）

Atop 在….顶上



已经存在了一个gray的内核，现在我们想在上面重叠一个带透明度的绿色WINDOW，会发生什么

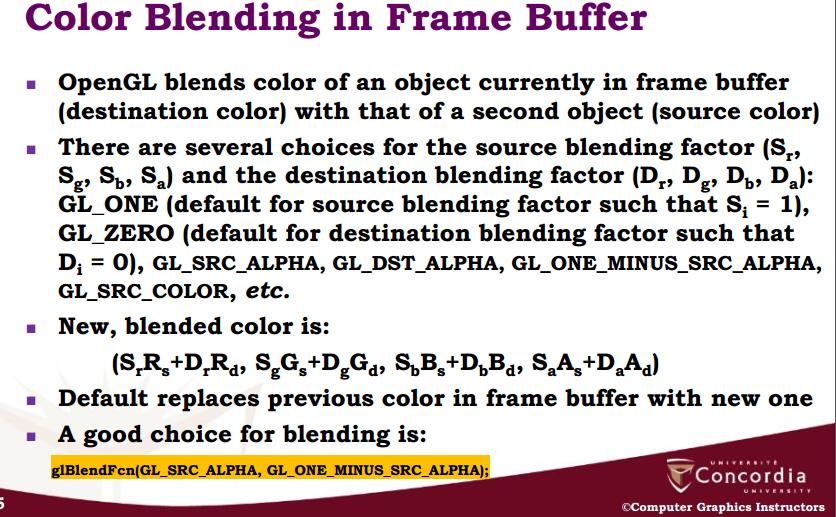
0,75是alpha value

低的就是透明

高的就是不透明

会发生blending，将新的color value 以及存在的color value混合，制造 半透明的场景

必须enable blending



注意new的是source

已经存在的是destination

GL BENDING有很多选择

GL\_ONE(默认source blending的blending factor 为1)

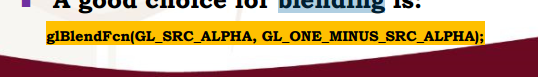
GL\_ZERO(destination的factor是0)等等

最后blended color



R G B是原始颜色

SR SG SB SA是对应系数（为什么要有系数，如果没系数，加起来很容易大于1，没意义）

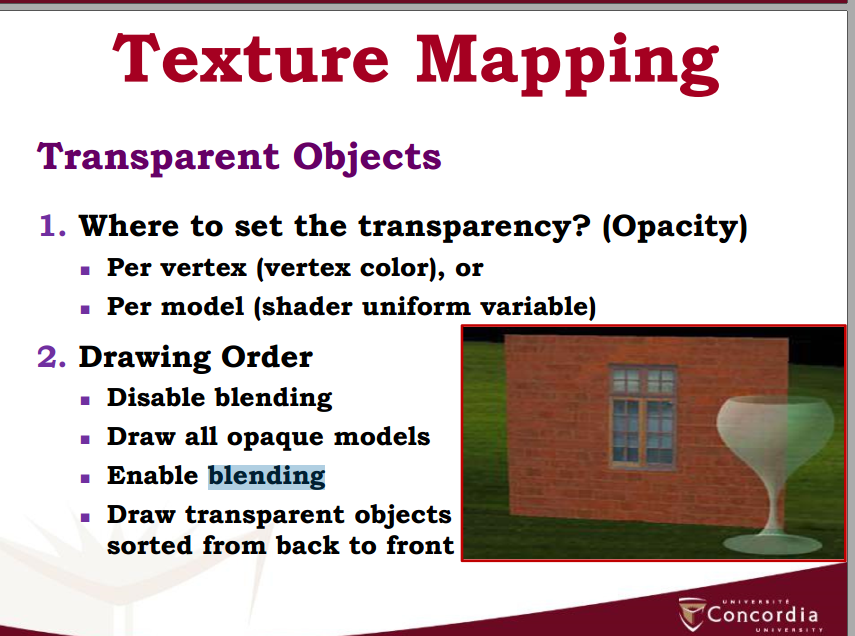


GL\_SRC\_FACTOR: GL\_SRC\_ALPH

GL\_D\_FACTOR:GL\_ONE\_MUNUS\_SRC\_ALPHA

解释：SRC是带有透明的东西，他的系数就是原有的alpha

被附着的系数就是1-SRC\_APHA



应该的绘画顺序

先禁止blending

然后画出所有的opaque model

然后允许blending

然后画出所有透明object，

从后到前

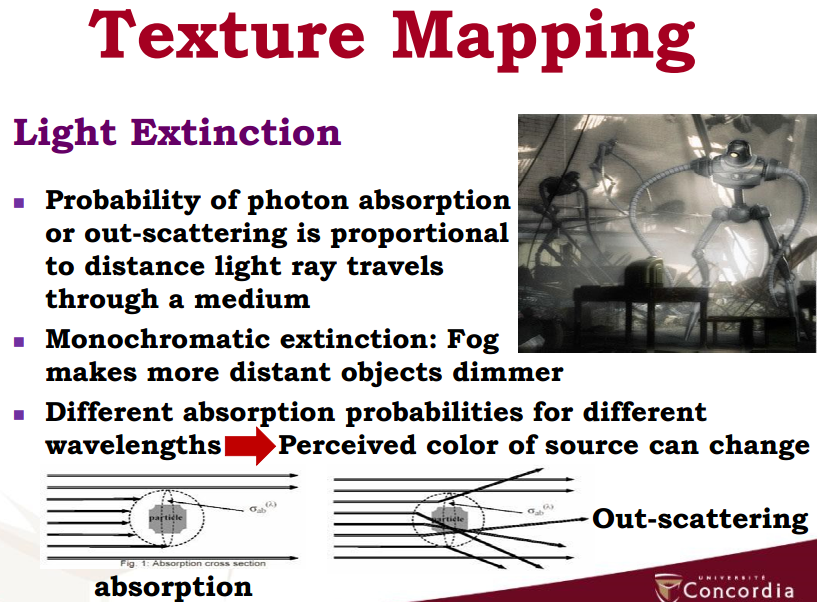
Light extinction消光

Photon光粒子

Out-scattering散射

Is proportional to成比例

Monochromatic,单色



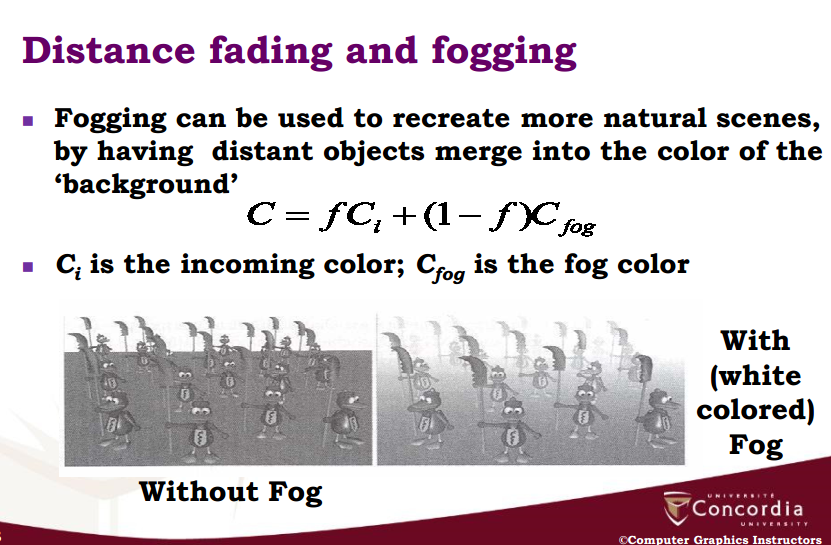
光通过一个介质medium 到了远处，，就有可能被吸收或者散射

雾让远处的物体变暗

不同波长的光被吸收的几率也不同

Fogging可以用来创造更加真实的场景

通过让远处的object与背景的颜色merge



F实际上是一个与物体 深度 有关的function



所以end指的是最远的线

Start指的是视野内最近的线

Z是所处depth

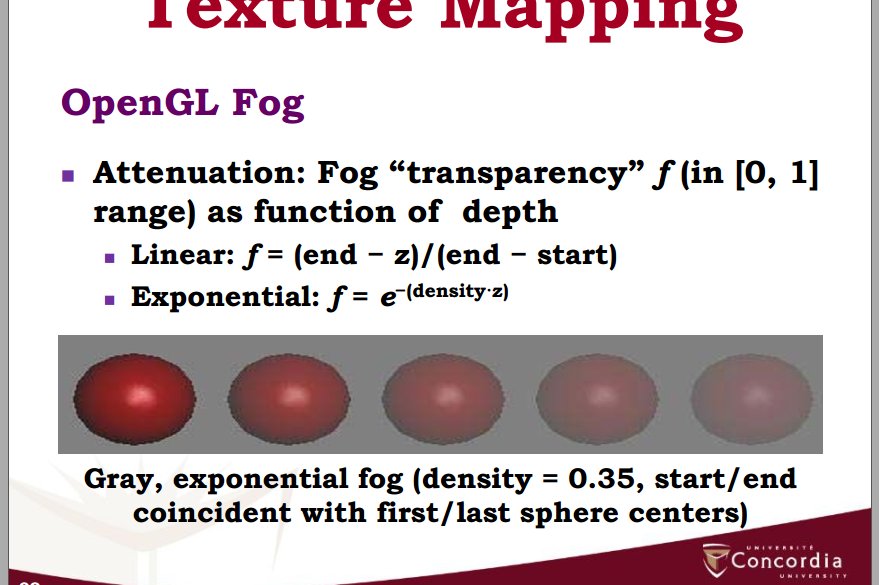
End-start是这个画面的纵深距离

End-z是这个人到end的距离

距离等于0即z=end， f为0，完全显示fog颜色

Z=start ,f=1,即开始的时候，完全显示本体颜色

Atenuation衰减，



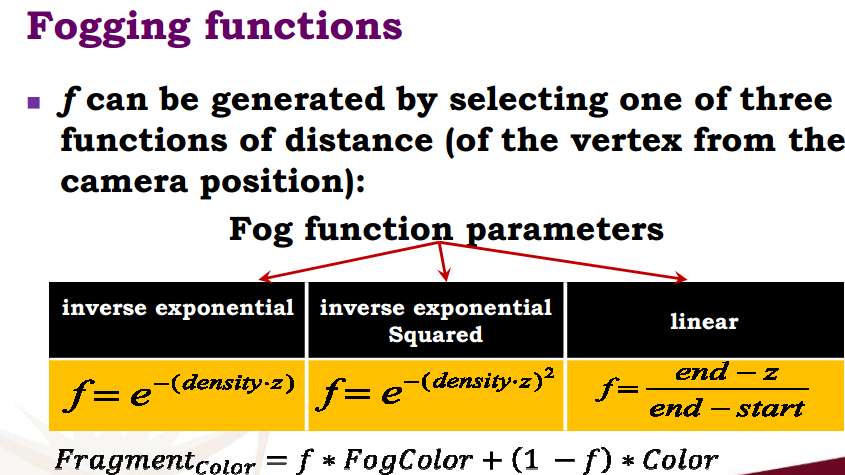
Opengl fog

小f有两种算法

而雾的透明度从0到1，在开头的时候是1，雾最不明显，结尾时0，雾最明显

线性算法的公式上一页有讲

还有指数算法，density密度， z， 也就是说 z 越大（越深）， f就越小（最后无限接近0）



三选一反正

但无论如何f的作用是一样的

开局是1，没雾，然后慢慢到0，雾拉满



Fog是由fragment shader实装的

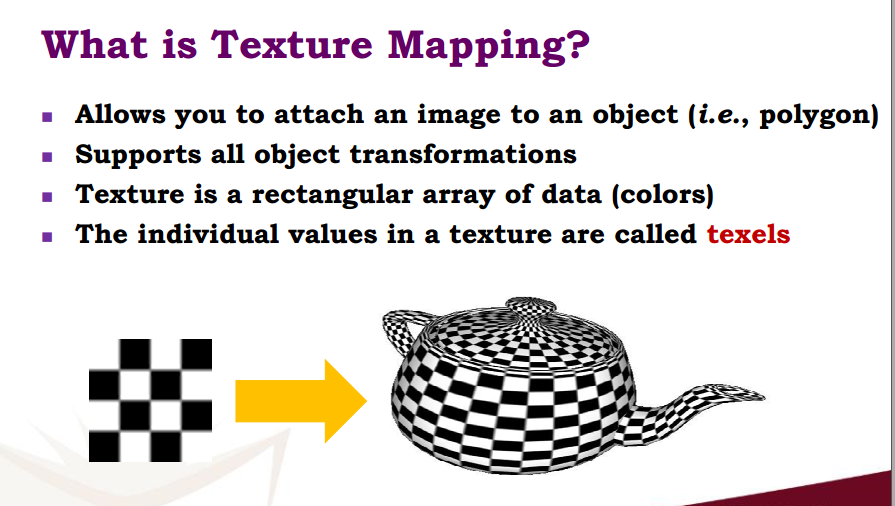
Texture mapping

允许你把一个Image图片附着到一个object上

支持所有object transformation（转动，变大，拉扯等等）

Texture实际上是一个矩形array of data (单纯是颜色)

Texture中的单个值被称作texels



Parametrically 参数的

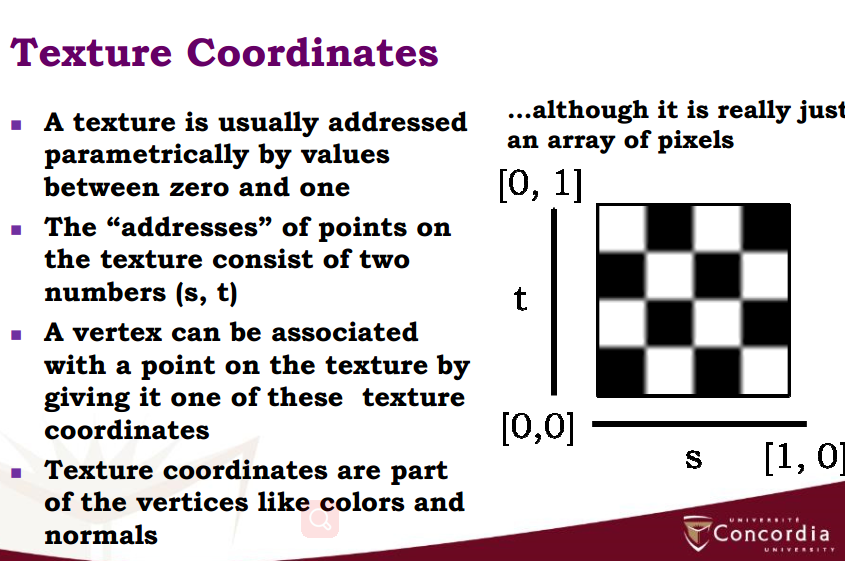
Texture Coordinates

参数上看，一个点的坐标是（s.t） s,t的范围都是 0到1

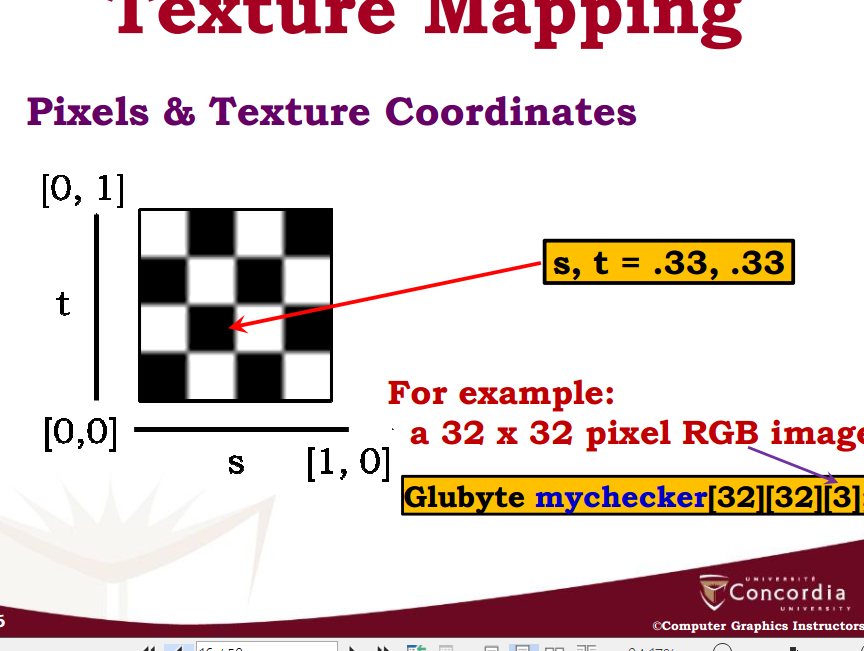
S是横坐标，t是纵坐标

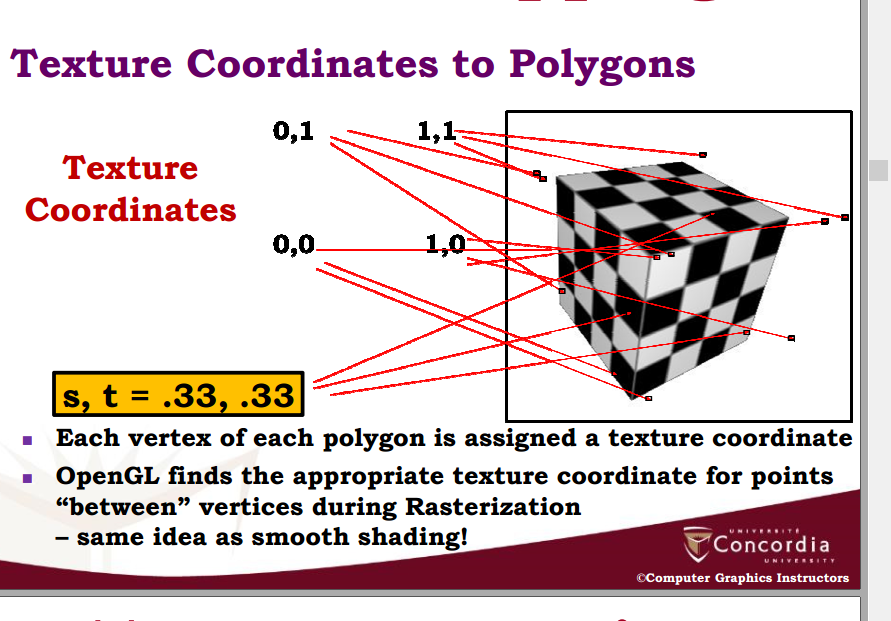
一个vertex可以与texture上的一个点的坐标所绑定

换句话说，texture上的coordinate对vertice来说，只是一个类似于color ,normal的vec2 数据



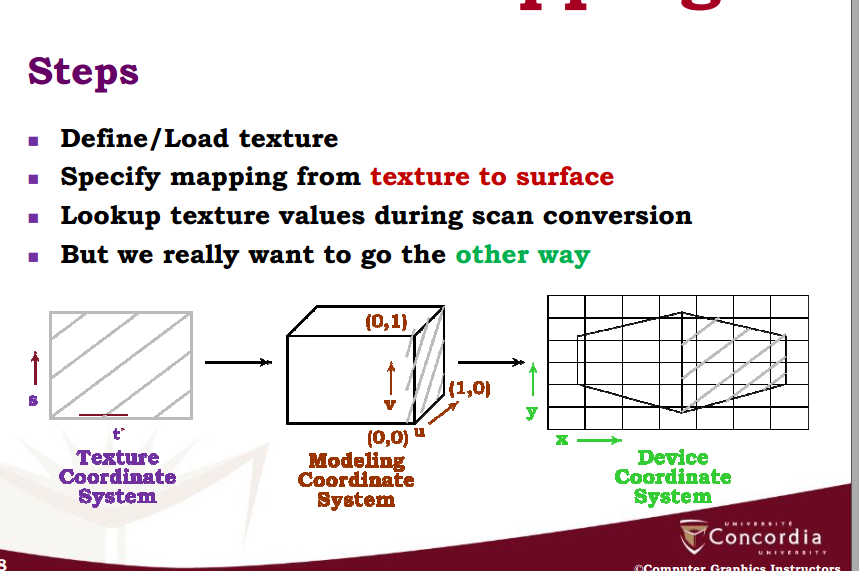
但是实际上这个矩形只是一个array，包含了像素点





实际上怎么运作的，每一个三角形都有texture coordinate数据， 然后OPENGL将这些三角形实际空间坐标与texture coordinate 坐标在rasterization阶段 一一对应， 这样就可以找到平面上的point了

lookup查找

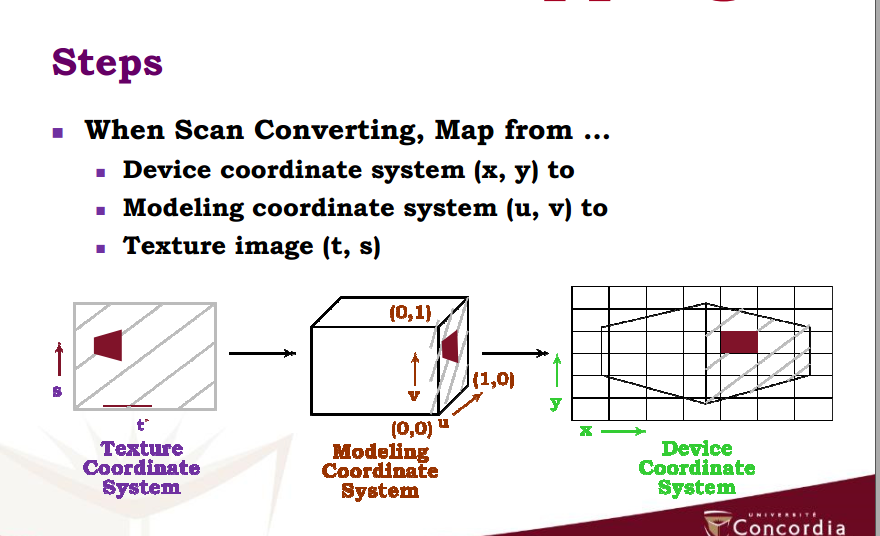


先load texture

把texture一一对应到surface上 （找到对应0 1 值）

然后再扫描转换的时候查找texture值 //这一步在下一页有具体描述

但我们实际上需要其他方法



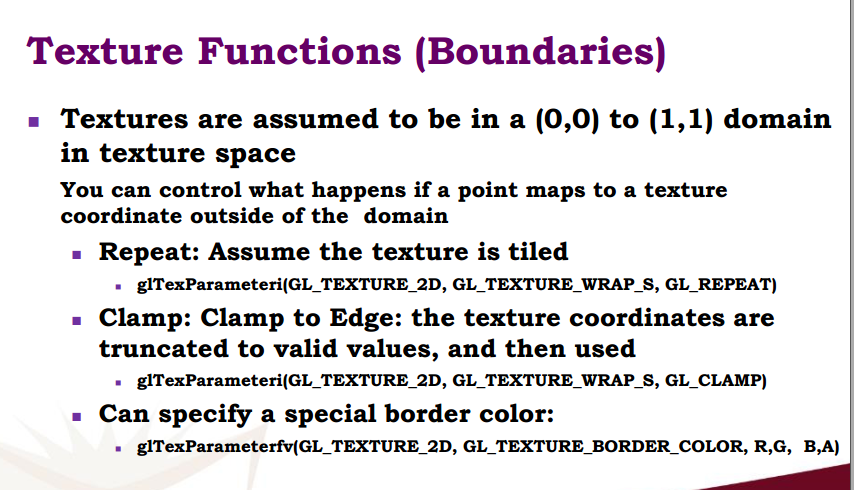
扫描转换的时候，

device的坐标（实际三维带变换）会去modeling的坐标（标准）

然后会去对应texture image

tiled平铺

clamp固定



texture自身的坐标在00到11之间

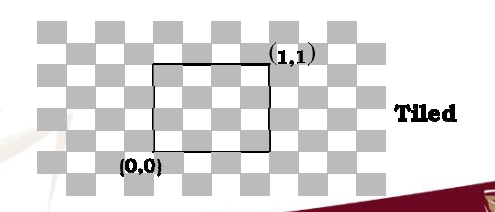
但是你可以控制如果一个模型的点map到了texture coordinate定义域以外的情况（指向2或3）

1.repeat:

假设这个texture是平铺的

使用

注意repeat关键词

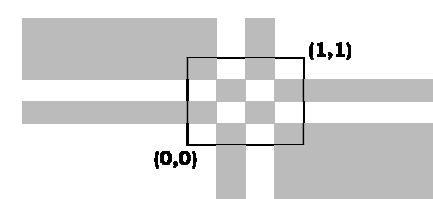


2.

clamp:固定

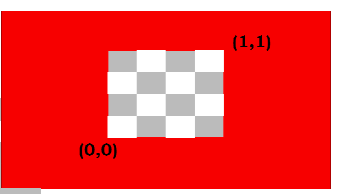
texture坐标就是0到1之间，然后上下左右无限重复





3.border color:你也可以指定0到1以外的颜色



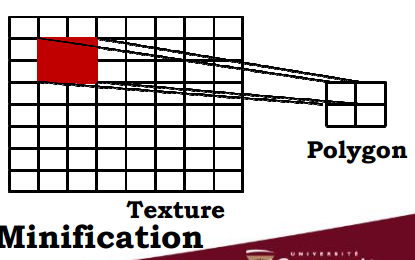


Magnification 与minification

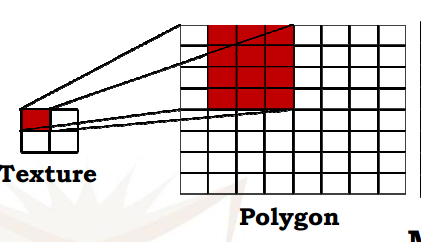
放大与缩小

minication:多个texture的像素texel对应一个模型的像素pixel.

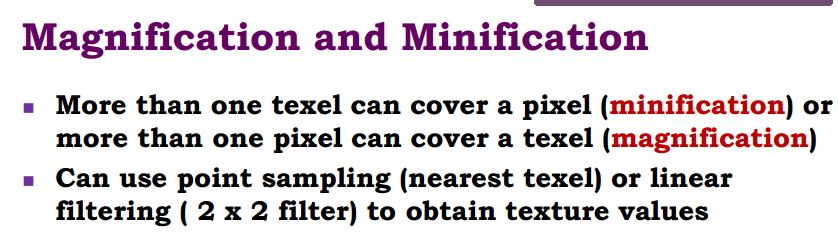




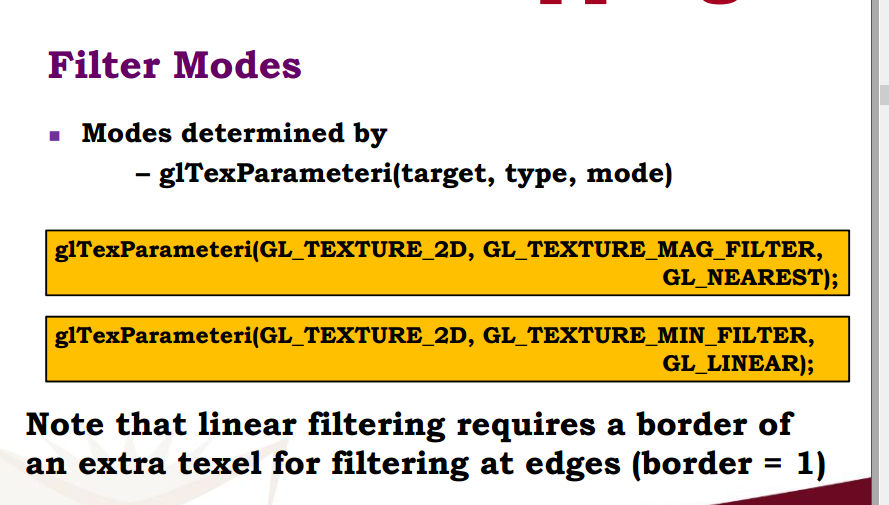
magnification:一个texture的texel对应多个pixel







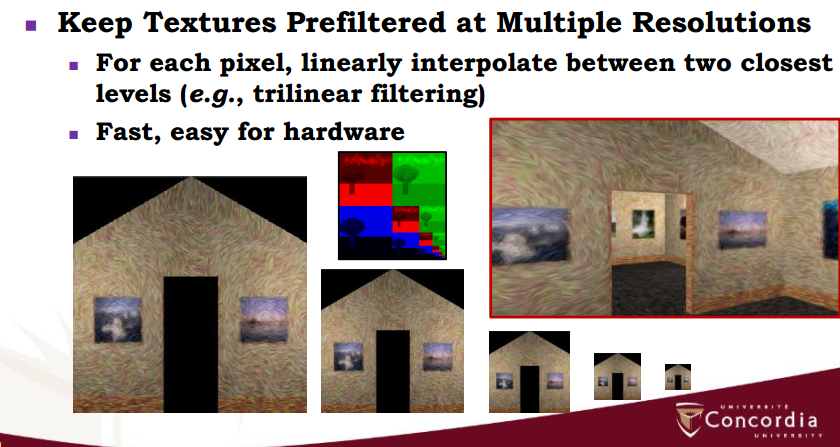
可以使用point sampling或linear filtering来得到texture值



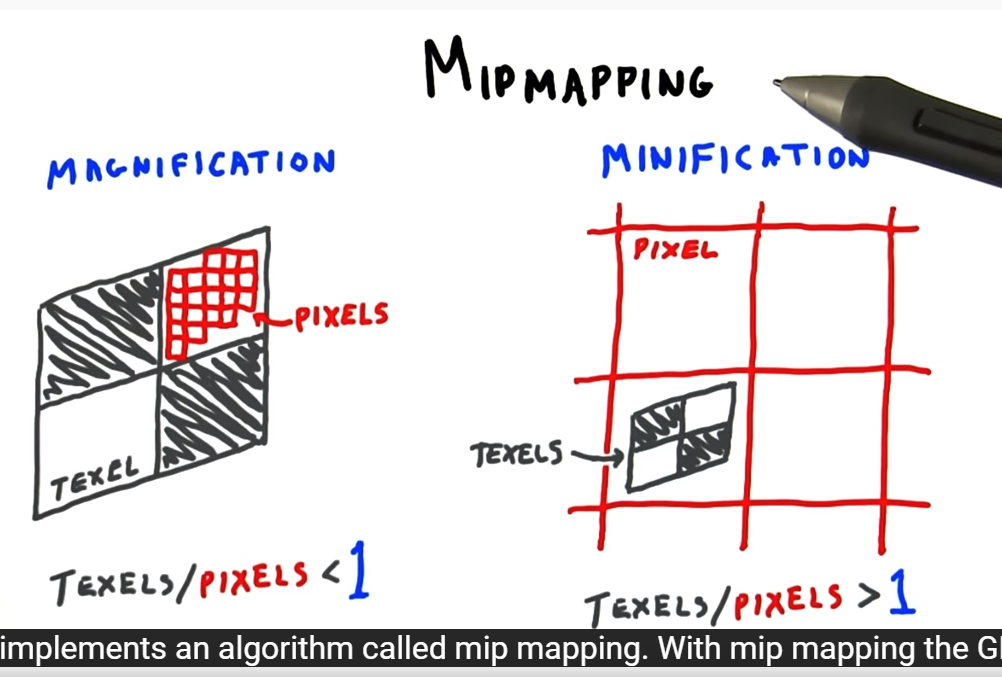
filter过滤

记住min filter还需要额外的texel来填充edge

mip maps 多级纹理



让texture根据texel与Pixel的比例，预先处理成不同的分辨率

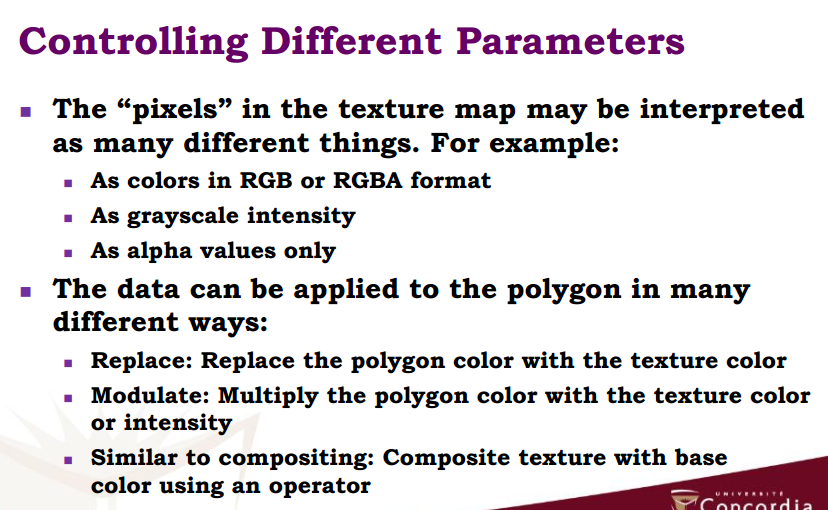


GPU分辨 texel与pixel的比率，texel比较少就magnification



通过这个比率预处理纹理，让他一开始分辨率就合适，texel越大（纹理分辨率越大），我们就缩小texture以应对合适模型

interpret描述



控制不同的参数

texture map的像素是由多个因素决定的

rgba

grayscale 灰度

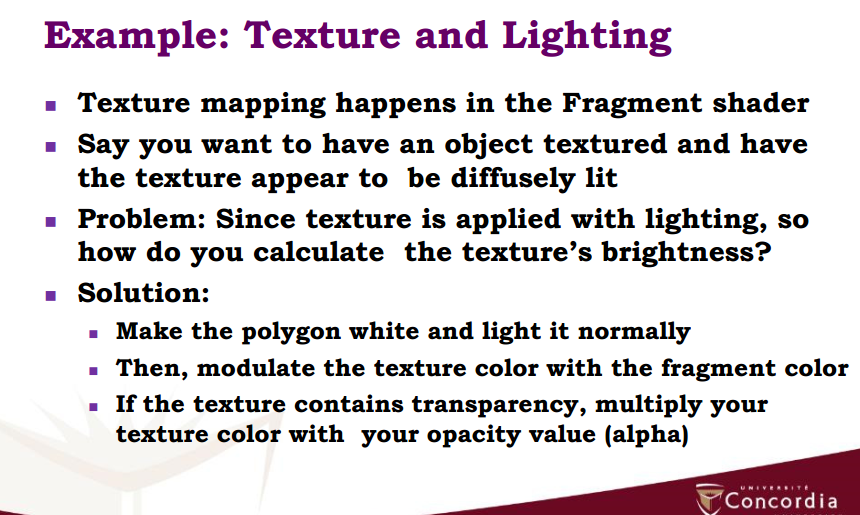
apha value 只有alpha

texture作用于polygon的方式

1.REPLACE:用texture color完全替代Polygon color

2.modulate:调节， 用polygon color与texture color相乘或将polygon color乘上一个intensity强度

3.compositing: 复合， 通过一个Operator将textuer与base color组合到一起



texture mapping在fragment shader中实现

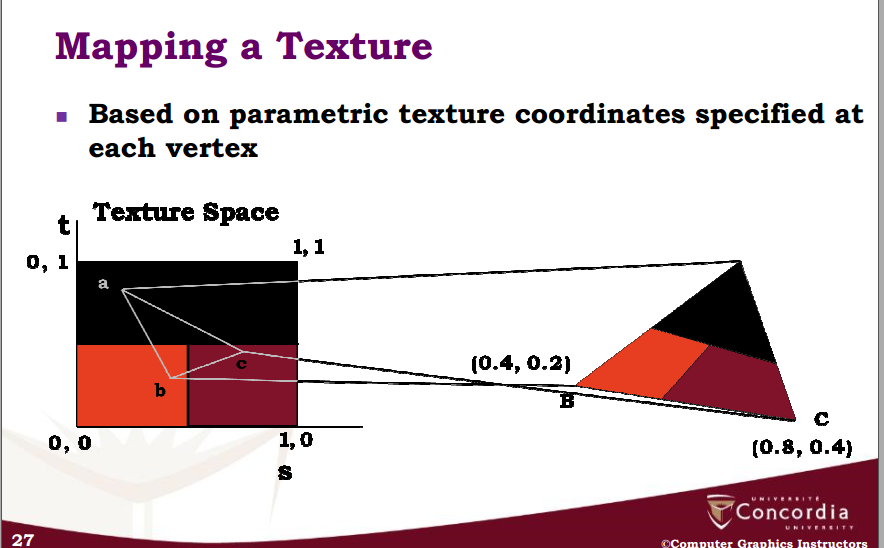
假设你想让一个object textured，并且漫反射

那么怎么决定texture的亮度呢?

1.让polygon变成白色，像普通一样照射

2。然后modulate，将texture color与fragment color相乘

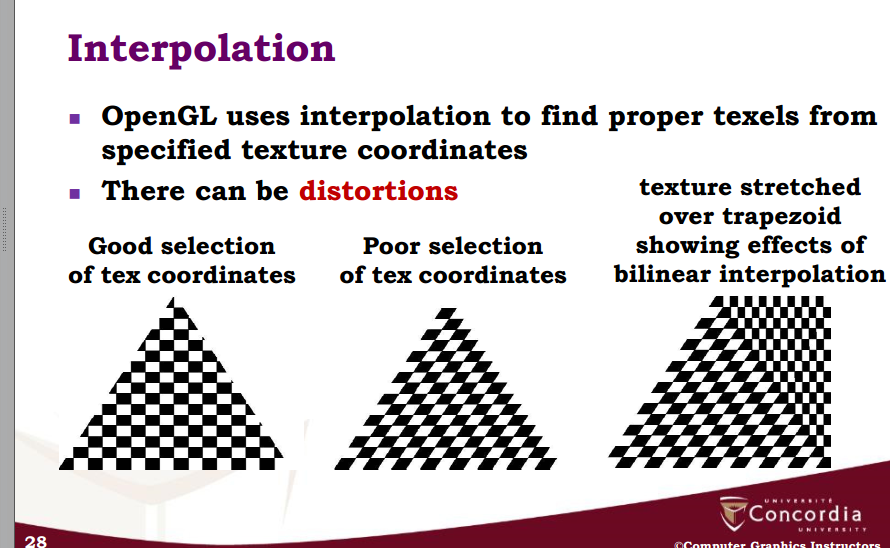
3.如果texture有透明属性，将texture color乘上 透明值



怎么将texture map到一个polygon上

通过vertex里所指定的texture coordinate 那个vec2参数

interpolation



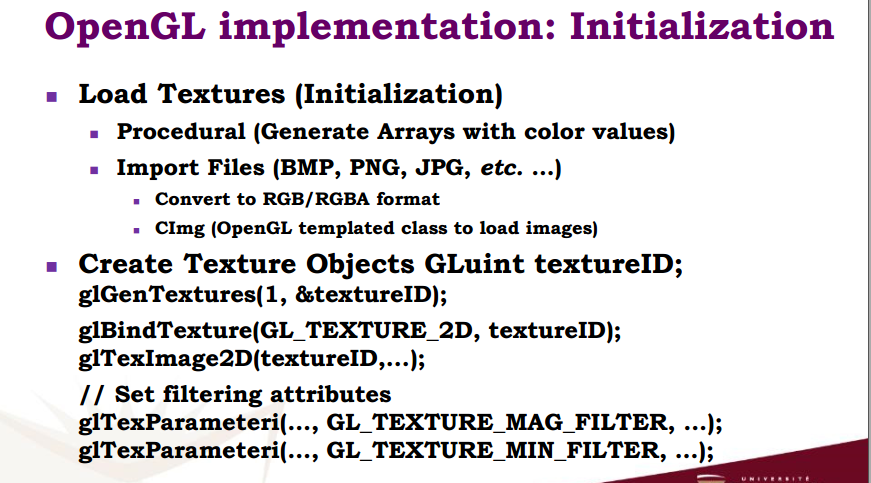
OPENGL 使用intepolation插入来寻找合适的texel

这有可能会distortion失真

tex coordinate定位准确就是左边

定位差（就是vertice的vec2没写好）就是右边

超出边界了就是最右边



怎么样初始化一个texture

1.initialization， load

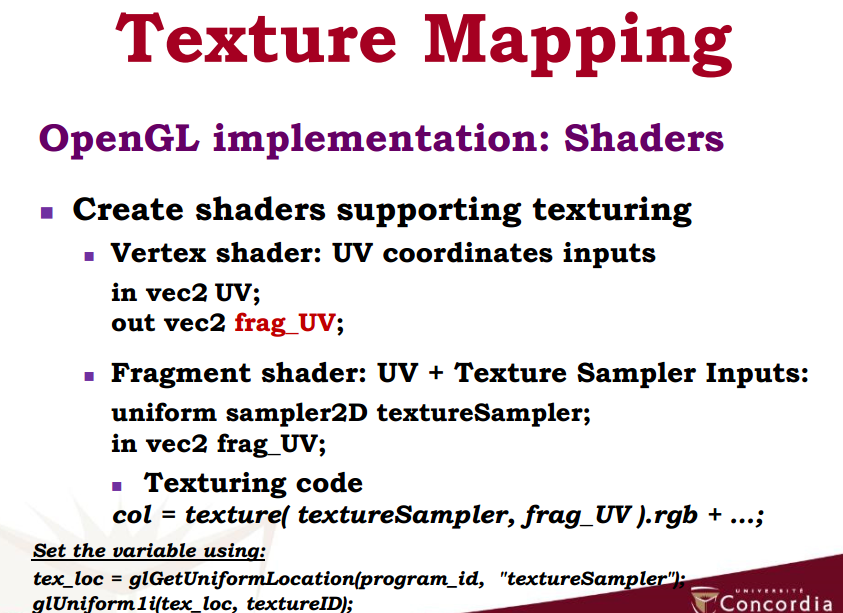
预处理procedural (生成一个要填充颜色值的array)

载入File，把他们转化成RGBA格式

2.创造textureID

生成ID，绑到一起，

设置filtering

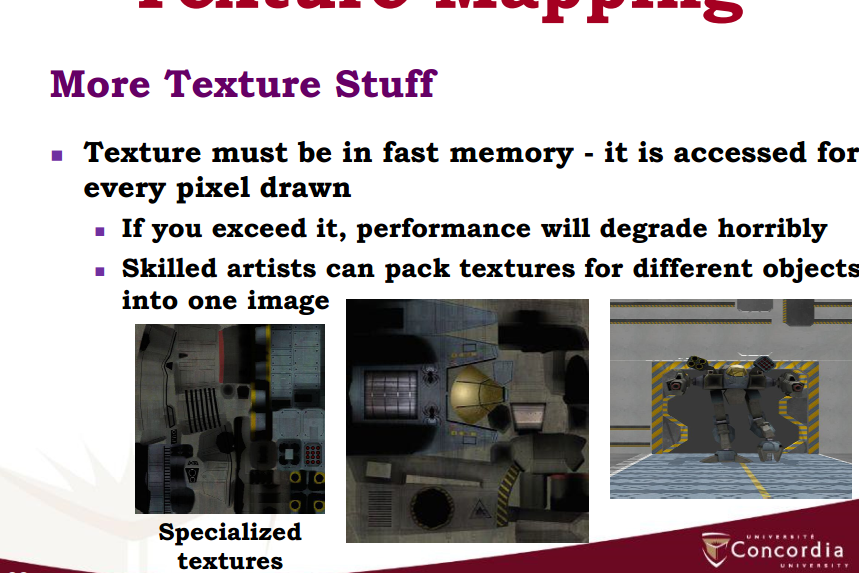


uv input vec2 coordinate //VEC2坐标

输出的是frag\_UV

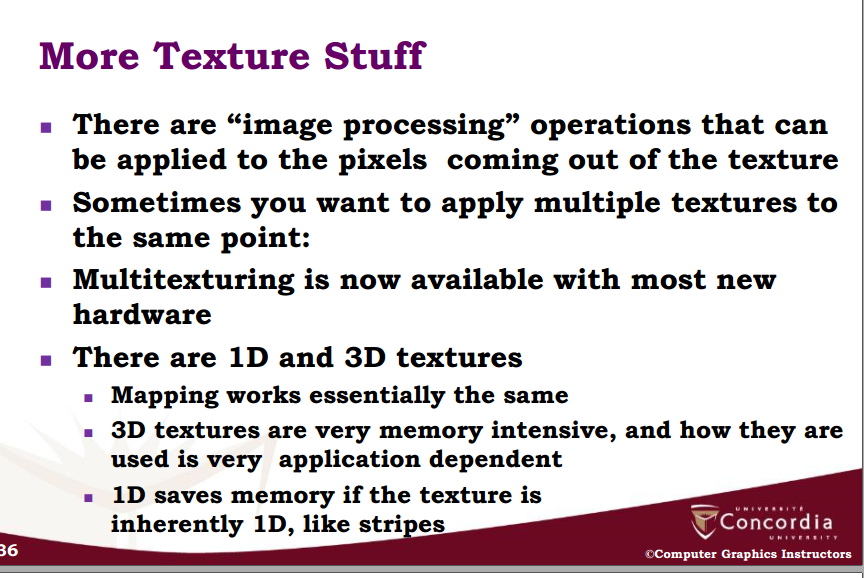
FRAGMENT SHADER:  
输入UV坐标

颜色等于texture颜色....



texture必须被存放在fast内存中

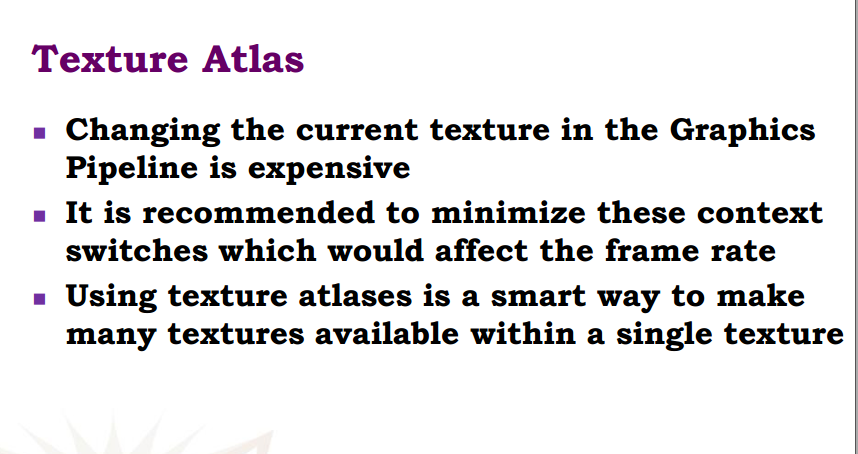
不然渲染会非常非常慢



multitexturing: 把多个texture覆盖到同一点上

1D texture:节省内存如果texture 天生是1D的，例如条纹

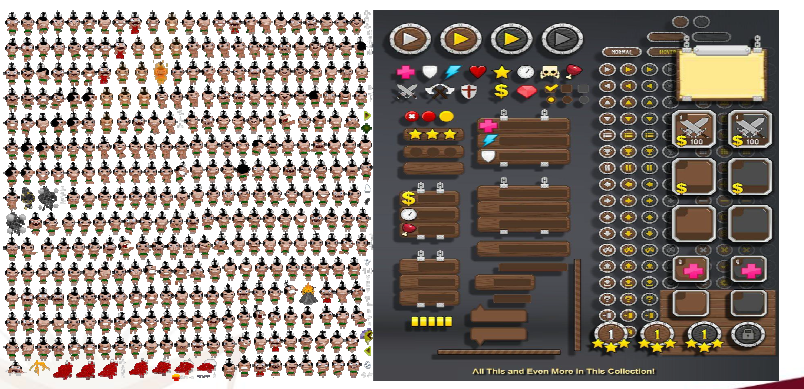
3D texture: 对内存敏感

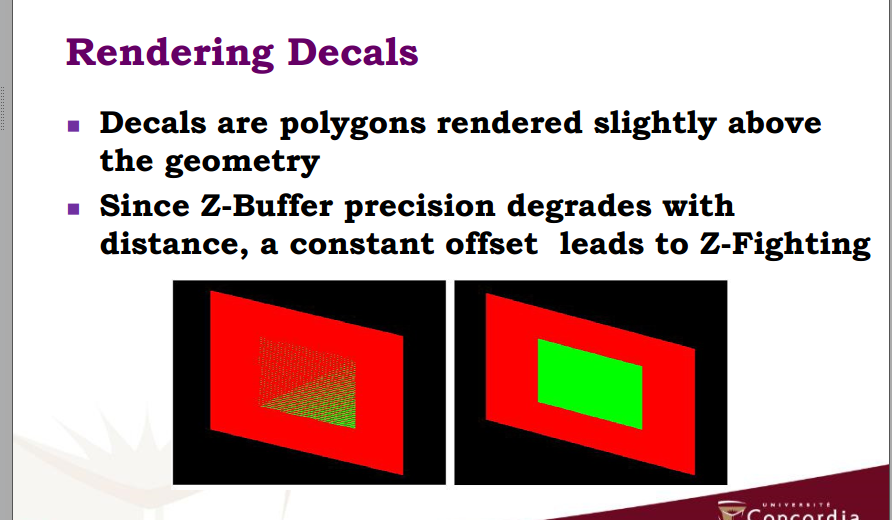


纹理地图集

减少texture的交换

所以实用texture atlases 纹理地图集 来吧许多纹理放在同一个纹理上

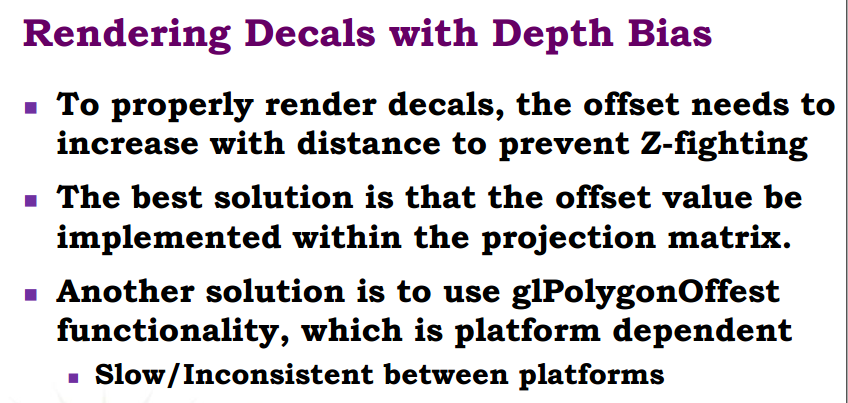




几何图形上方的多边形

因为z-buffer 的精度随着distance降低 , 这时恒定的z抵消 offset会导致z-fighting

解决方案

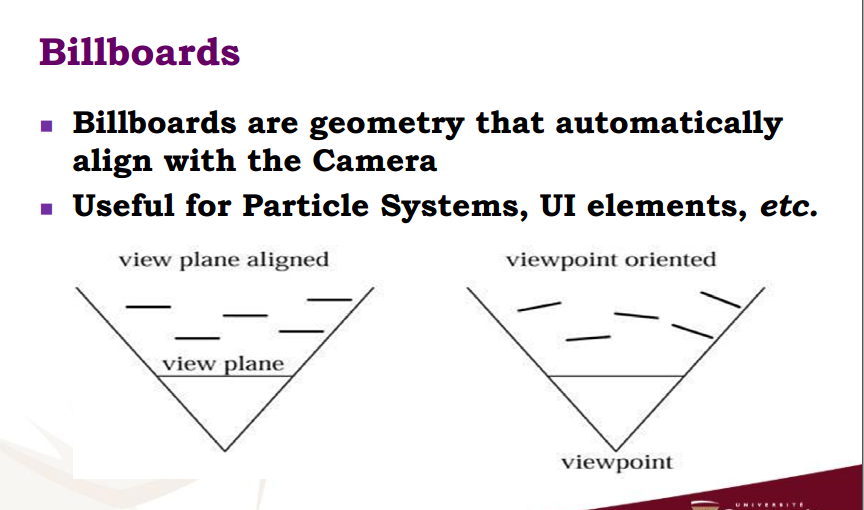


increase offset

随着distance的增长，增加offset(通过使用projection matrix)

或者使用glPolygonOFFset

billboard

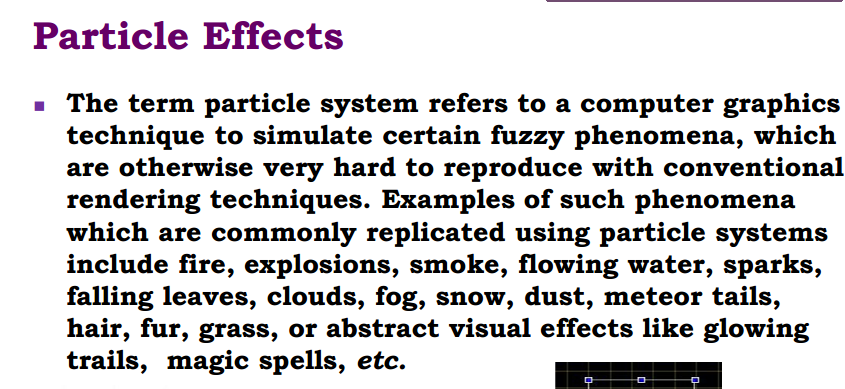


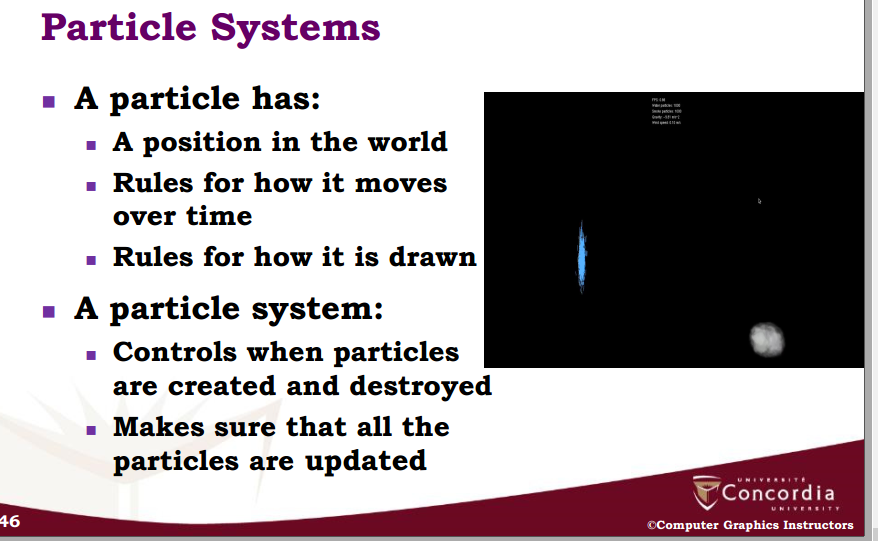
billboard是一种图形，会自动与摄像机对齐

左边是对齐的

particle effect 粒子特效

粒子系统是一种电脑图形科技,用来模拟那些fuzzy模糊的现象，使用传统方法很难重现，例子比如说火焰，烟雾等等





一个粒子具有

世界中的一个坐标Position

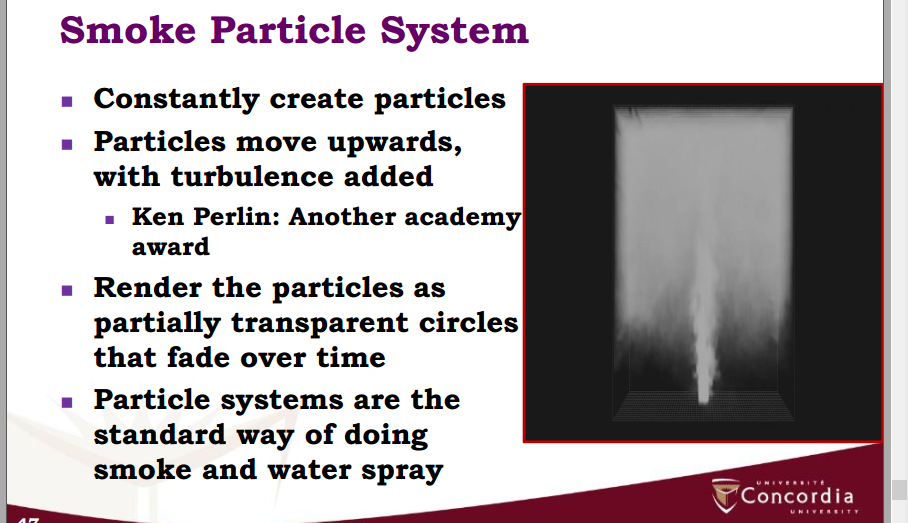
随着时间移动的具体法则

怎样画出来的规则

一个例子系统具有

控制什么时候创造粒子，消灭粒子

确保所有粒子都在更新



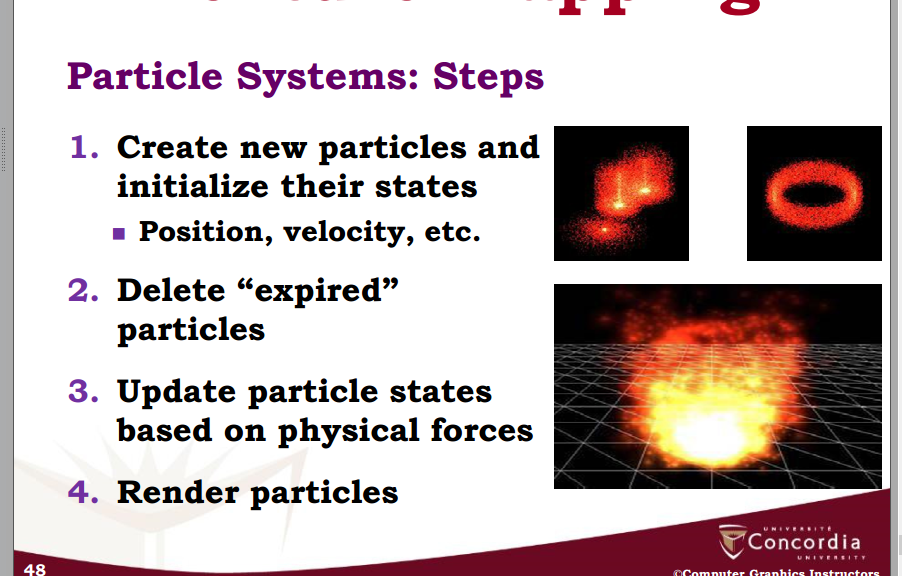
烟雾粒子系统

持续产生粒子

粒子往上，并且加上turbulence骚乱

把粒子花城一个部分透明的圆，并随着时间消逝

粒子系统是标准方法来模拟烟雾以及水蒸气

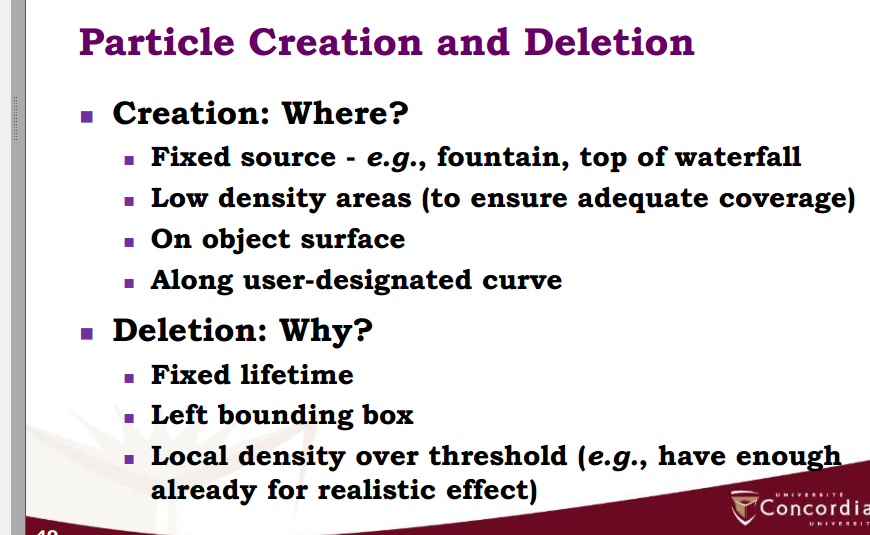


创造新的Particle并且初始化他们的state，例如位置速度等

删除过期的particle

基于物理现象更新这些particle状态

渲染particle



创造地点

是固定的，例如喷泉

低密度地区低密度地区来确保适当的覆盖

在物体表面

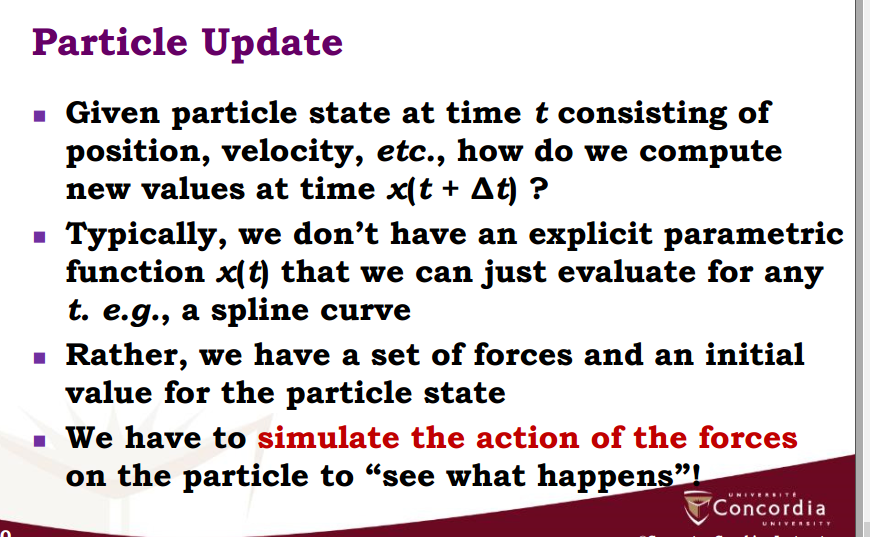
沿着user指定的曲线

删除

时间固定

粒子效果范围固定

当前密度过高



particle update

现在有一个particle包括了位置，速度，那么我们怎么计算下一个时间点的新值呢

通常，我们没有一个明显的参数方程来计算任何值

我们只能给他一组力，以及初始参数来模拟物理系统

然后模仿力作用在粒子上，看看会发生什么