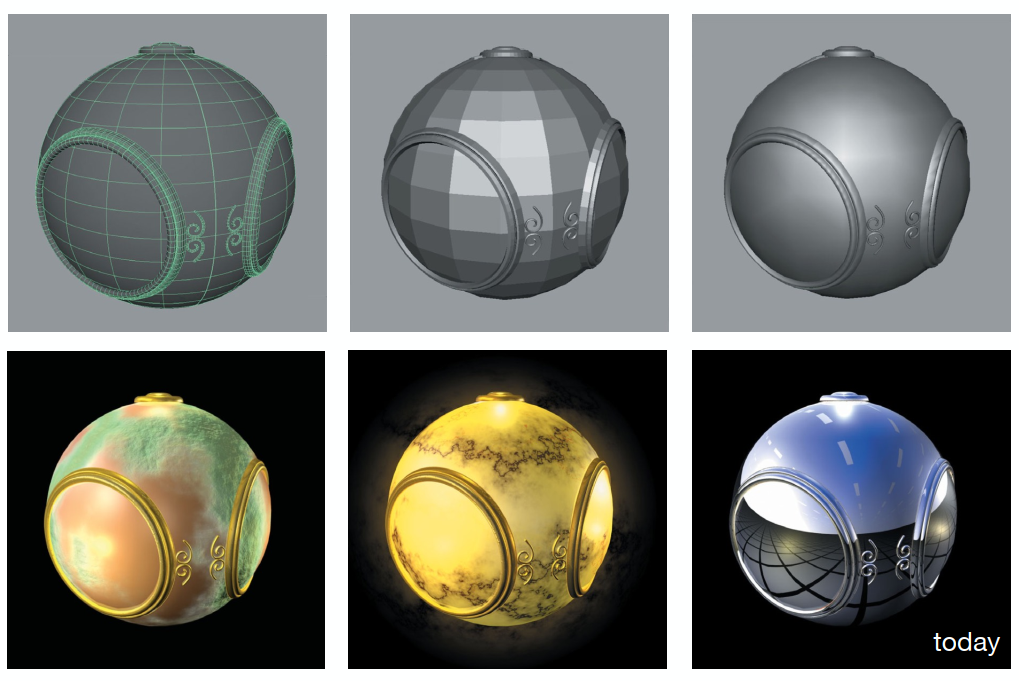
# (四)Applications of Textures

纹理的用途非常多



In modern GPUs, 纹理 = 内存+ 范围查询 (filtering)

Environment lighting

Store microgeometry

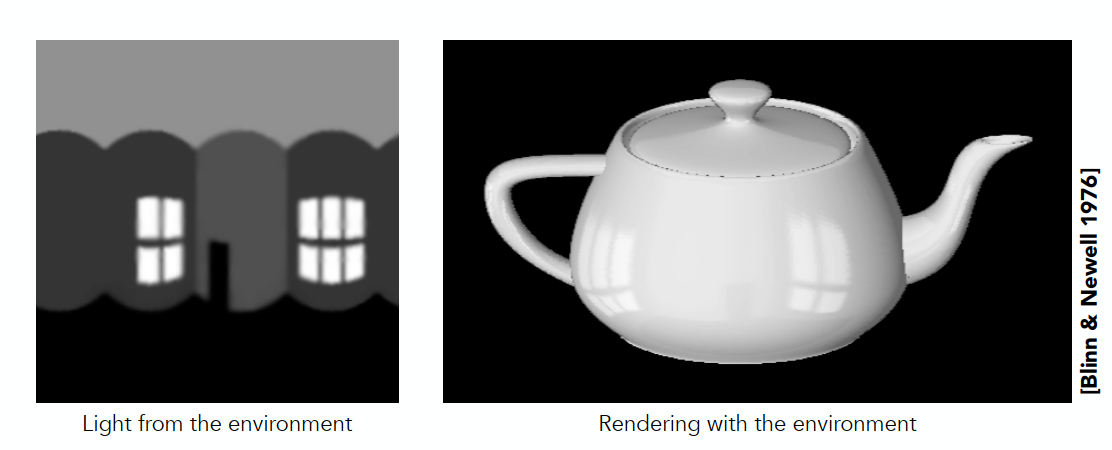
Procedural textures

Solid modeling

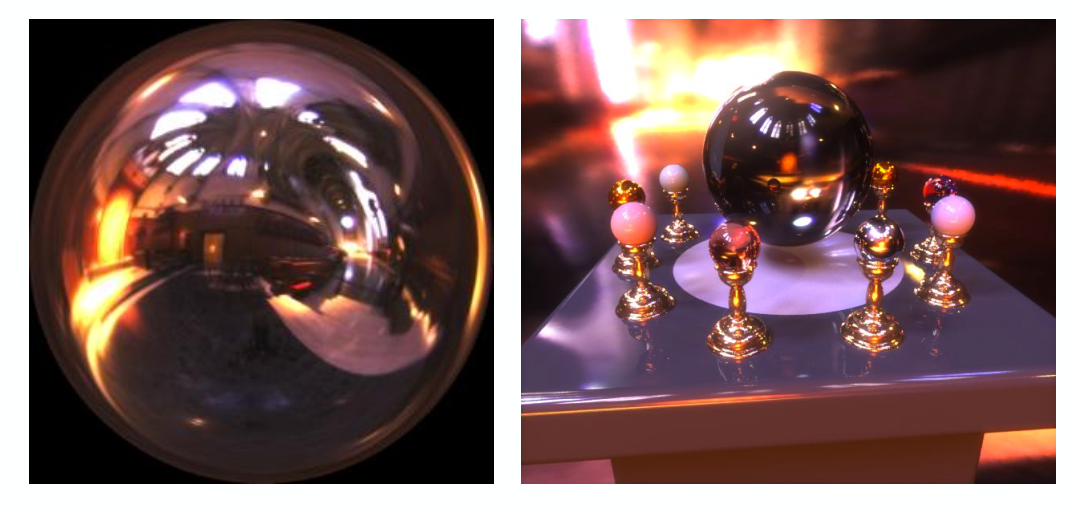
Volume rendering

…

## 1、Environment Map

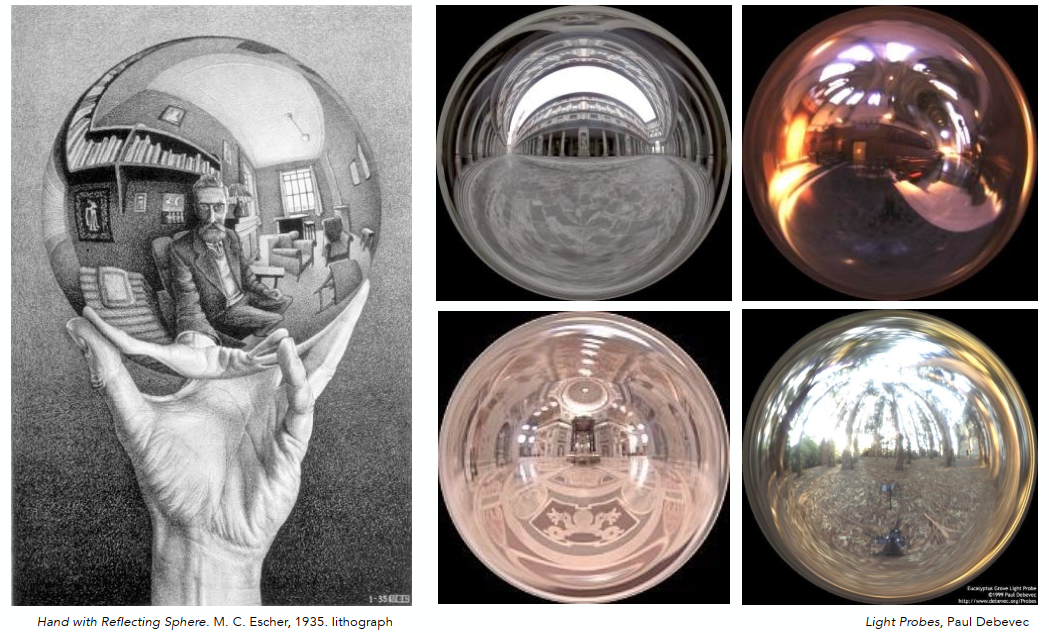


环境贴图，制造环境光，描述来自不同方向的光照信息，照亮整个环境中的物体，也可以被环境中的物体反射。环境光只记录光的方向，即假设环境光来自无限远处，从同一个方向照过来的光强度都是一样的。(如上图就是拿左图环境光去渲染右侧茶壶)



左侧的这个环境光球如何去看？可以假设拿了一个非常光滑的金属球（镜子球），这个球反射出来了周围的环境。

### Spherical Environment Map



可以把整个环境光记录在球上，用球去存储环境光的信息。

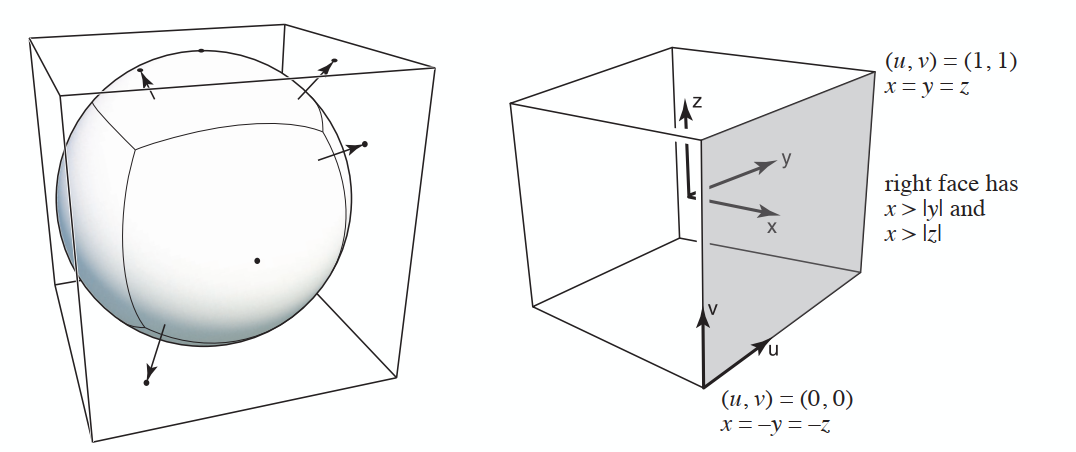
### Spherical Map — Problem



Prone to distortion(容易变形) (top and bottom parts)!

但是我们来看一个环境光球展开后的顶部和底部，可以发现出现了不同程度的变形。

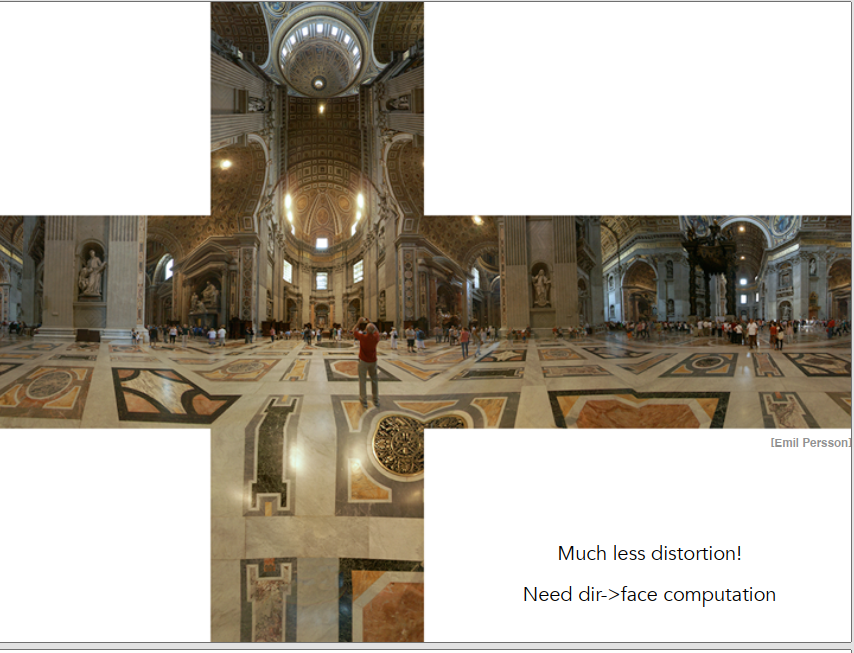
### Cube Map



A vector maps to cube point along that direction.

The cube is textured with 6 square texture maps.

用一个立方体盒子，把这个球包围起来。当光从中心打到包围的球上时，我们不管，让光继续走，直到碰到立方体的某一个面，记录下来信息。



那么展开后的图像有六个面，就不存在刚刚球展开后产生的大面积扭曲了。

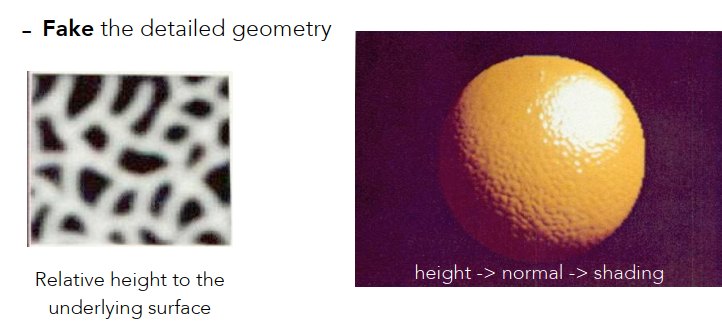
## 2、Textures can affect shading(凹凸贴图)

Textures doesn’t have to only represent colors(纹理不必只表示颜色)

What if it stores the height / normal?（如果它存储高度/法线怎么样？）

Bump / normal mapping(凹凸/法线映射)

Fake the detailed geometry(伪造详细的几何形状)



贴图不光可以表示颜色信息，还可以表示高度信息。高度贴图可以定义任意一个点，沿着他的法线向上向下走的相对高度。如右图，原本就是一个很普通的球，可以用很少的三角形来表示，如果想通过直接改变模型的方法得到右边那种凹凸不平的效果，会用很多三角形，开销会很大，那么在这里通过凹凸（高度）贴图，改变高度信息，高度信息变化，就会使法线信息变化，从而使着色结果发生变化（人们看到的明暗对比一定程度上就是因为法线变化），得到右图的效果。

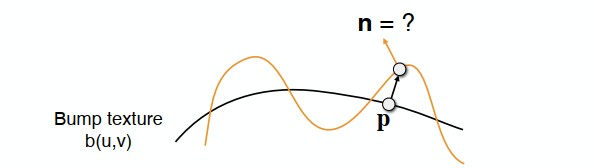
### Bump Mapping(凹凸映射)

Adding surface detail without adding more triangles在不添加更多三角形的情况下添加曲面细节

Perturb surface normal per pixel (for shading computations only) 每个像素法线的扰动表面（仅用于着色计算）

“Height shift” per texel defined by a texture由纹理定义的每个纹素的“高度偏移”

How to modify normal vector? 如何修改法向量？



如上图，假设黑色的线是原本一个比较光滑的面，在这里应用了一个凹凸贴图，凹凸贴图告诉我们一个相对高度如何变化，得到了黄色的线，那么原来点的位置会认为被凹凸贴图改变，那么法线也会改变(原本的法线是p，改变后法线是n)。

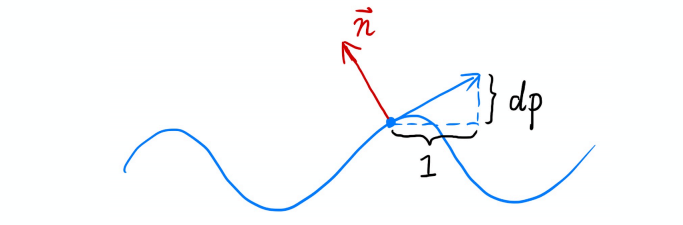
凹凸贴图改变相对高度，从而改变法线。

### How to perturb the normal (in 1D)

原始表面在p点的法线:n(p) = (0, 1)

P点处的导数(切线) :dp = c \* [h(p+1) – h(p)]

改变后的在p点的法线:n(p) = (-dp, 1).normalized() (切线逆时针旋转90度并归一化)



我们认为，某点p原来的法线n为(0, 1) ，我们要求出改变后的法线，首先就要求出切线，设切向量为（x，y）

由于点p内存储着这一点的高度信息，因此会改变p点的高度，为h（p），利用微分的思想，我们再去找相邻像素点的位置，即p+1处，也有一个高度h（p+1），通过这两点的高度差，就可以算出切向量的y值，即为dp，x的值就是相邻两个像素的x，为1，因此切向量即为（1，dp），所以法向量就是（-dp，1），即法线为（-dp，1）。

### How to perturb the normal (in 3D)

原始表面在p点的法线: n(p) = (0, 0, 1)

P点处的导数(切线) :

​ --dp/du = c1 \* [h(u+1) - h(u)]

​ --dp/dv = c2 \* [h(v+1) - h(v)]

改变后的在p点的法线:n = (-dp/du, -dp/dv, 1).normalized()

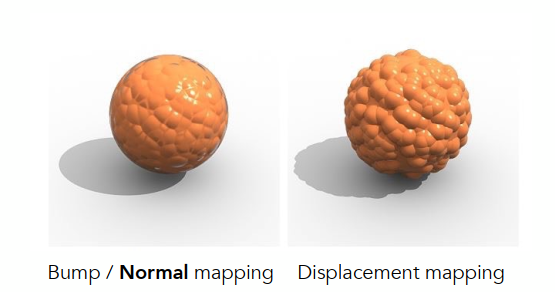
Note that this is in local coordinate

将刚刚一维上一条线的思维推广到二维上uv平面内的操作，我们即可得出上述的公式。

### Displacement mapping(位移贴图) — a more advanced approach

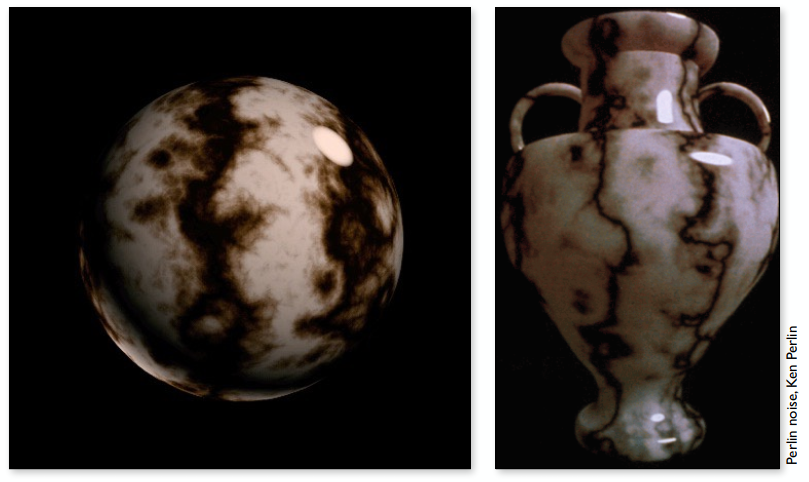
Uses the same texture as in bumping mapping

Actually moves the vertices



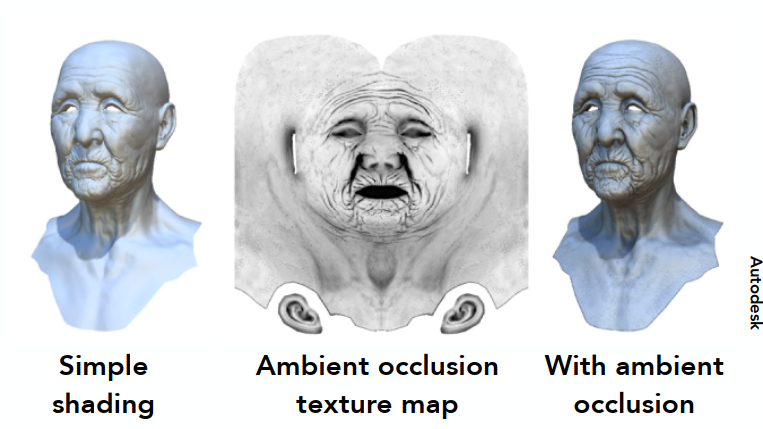
如上图可以看出，左侧通过法线贴图改变模型表面细节对于边界和阴影是有破绽的，而右侧的位移贴图，直接通过贴图改变模型各个三角形的顶点位置。但是位移贴图要求三角形数量足够多，否则精度会非常低。

### 3D Procedural Noise + Solid Modeling



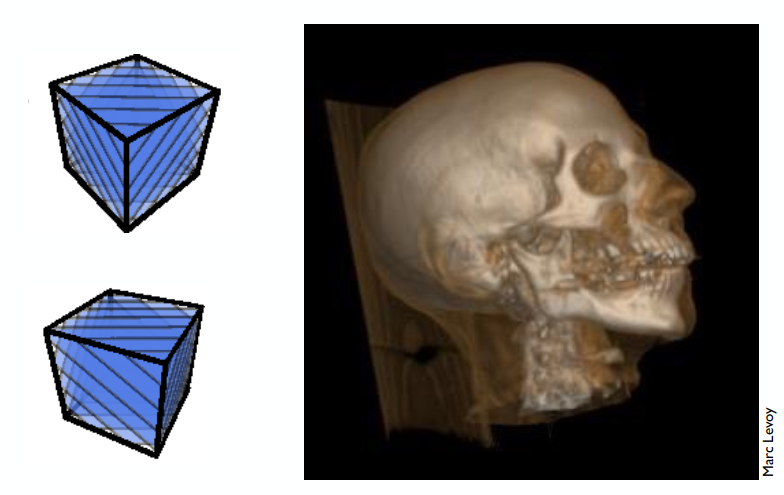
除了二维纹理，还可以定义三维纹理，如果把这个球砍一半，可以看到其内部的纹理，这里实际定义了空间中任何一点的值，这种纹理实际没有真的生成纹理的图片，而是定义了一个在三维空间中的噪声函数，对于空间中任意一点都有一个解析式可以算出在该点的值。

### Provide Precomputed Shading



纹理贴图还可以记录一些提前算好的信息，比如模型自己本身互相遮挡产生的阴影（环境光遮蔽）。

### 3D Textures and Volume Rendering



核磁共振返回每个三维点的信息，可以用作体积渲染。存储为3维纹理

# Many Ways to Represent Geometry

Implicit(隐式几何)

algebraic surface

level sets

distance functions

…

Explicit(显式几何)

point cloud

polygon mesh

subdivision, NURBS

…

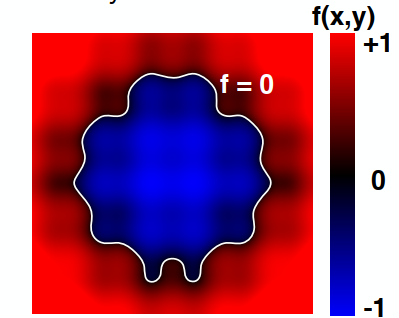
Each choice best suited to a different task/type of geometry

# (一)“Implicit” Representations of Geometry

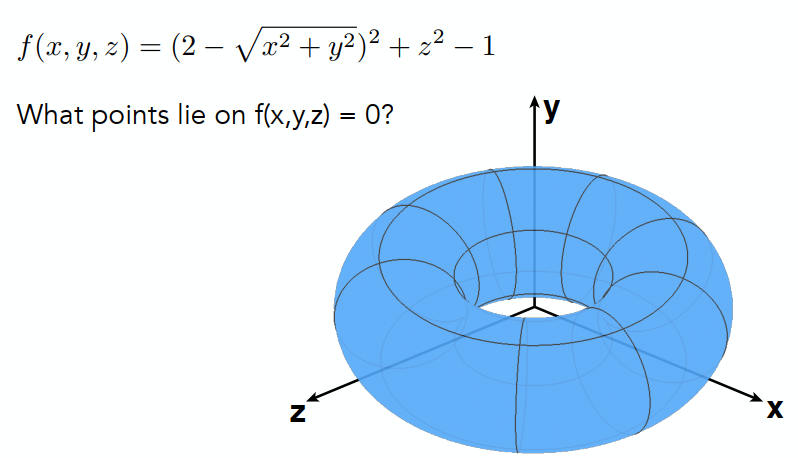
## 1、What is implicit geometry

隐式几何：表示一定的关系，并不给你实际的点（比如一个球的函数x²+y²+z² = 1）

更通用来讲，定义一个函数f(x,y,z) = 0，只要满足这个式子，就认为这个（x，y，z）是我定义的表面上的一个点，如果能找到所有点就能把这个表面画出来。（如下图）



## 2、Implicit Surface – Sampling Can Be Hard(隐式坏处)



隐式几何存在一些问题，如上面的式子，如果一个点满足以上式子的关系，则认为这个点在这个函数所表示的几何的面上，但是如果去求哪些点满足这样一个式子，是相对复杂的问题。虽然我们能直接画出来，是一个圆环的结构，但是单单看式子，很难看出。也就是说隐式几何很难看出函数所表示的几何的“真容”。

## 3、Implicit Surface – Inside/Outside Tests Easy(隐式好处)

f(x, y, z) = x²+y²+z²-1 Is (3/4, 1/2, 1/4) inside?

Just plug it in:f(x,y,z) = –1/8 < 0

Yes, inside.

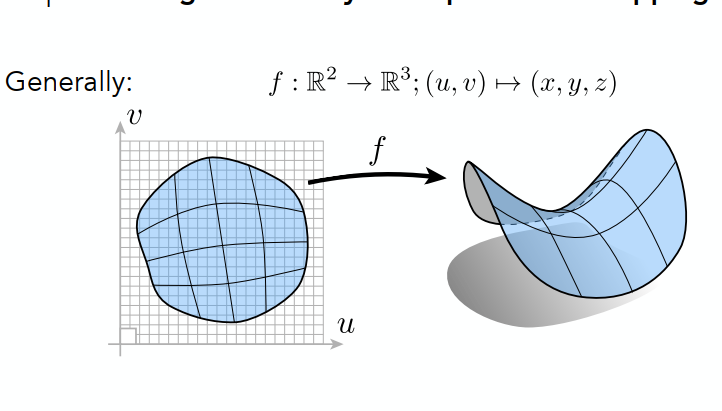
隐式几何也有一些好处，如上面的问题，判断任何一个点和几何的相对位置关系，只需将这个点带入隐式几何的函数中即可判断。如果得到的结果是正数，则说明点在几何外；如果得到的结果是0，则说明点在几何上；如果得到的结果是负数，则说明点在几何内。隐式的表示可以很容易让我们判断一个点在几何内还是外。

# (二)“Explicit” Representations of Geometry

## 1、What is explicit geometry

All points are given directly or via parameter mapping

显式几何：把所有点都表示出来，如三角形，把面上的点确确实实都表示出来。



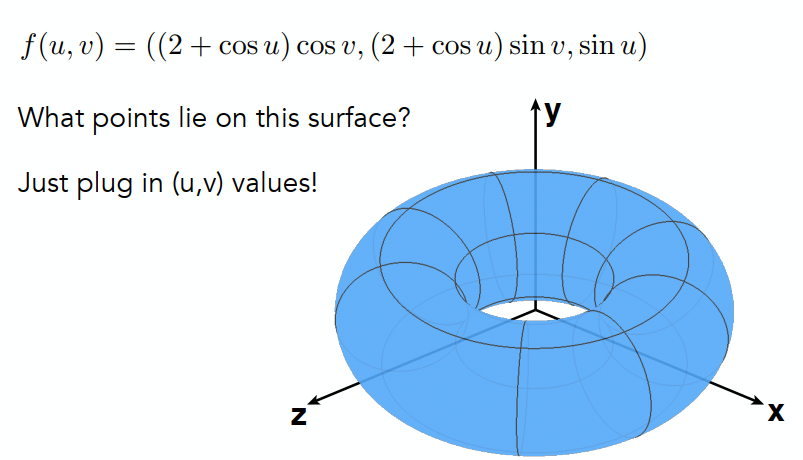
另一种显式的方法是通过参数映射。如上图，定义一个uv空间，上面有任意一个点用坐标uv表示，对应每一个uv值都可以映射到三维空间上的某一个点，把uv上所有的点都遍历一遍，就可以得到所有点的xyz，在三维空间中得到一个完整的几何体。

## 2、Explicit Surface – Sampling Is Easy（显式好处）

f(u, v) = ((2 + cos u) cos v,(2 + cos u) sin v,sin u)(参数映射)

What points lie on this surface?

Just plug in (u,v) values!



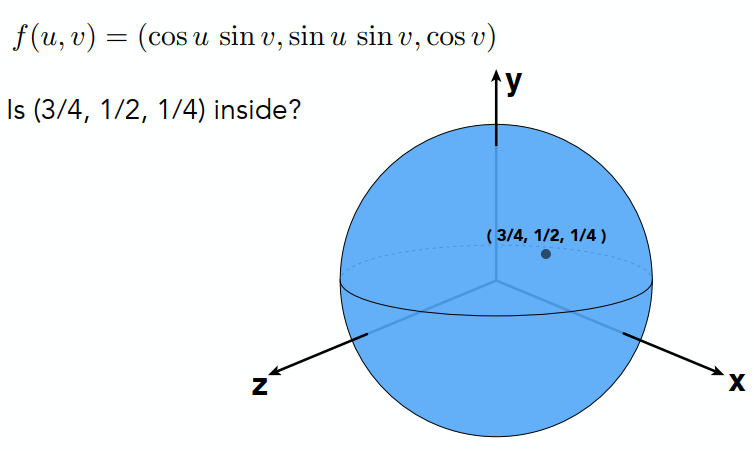
Explicit representations make some tasks easy

通过uv的参数映射，我们可以得到三维空间中所有点的xyz，得到这样一个几何体，因此显式几何可以很容易地看出几何图形的“真容”

## 3、Explicit Surface – Inside/Outside Test Hard(显式坏处)

f(u, v) = (cos u sin v,sin u sin v, cos v) Is (3/4, 1/2, 1/4) inside?

Some tasks are hard with explicit representations



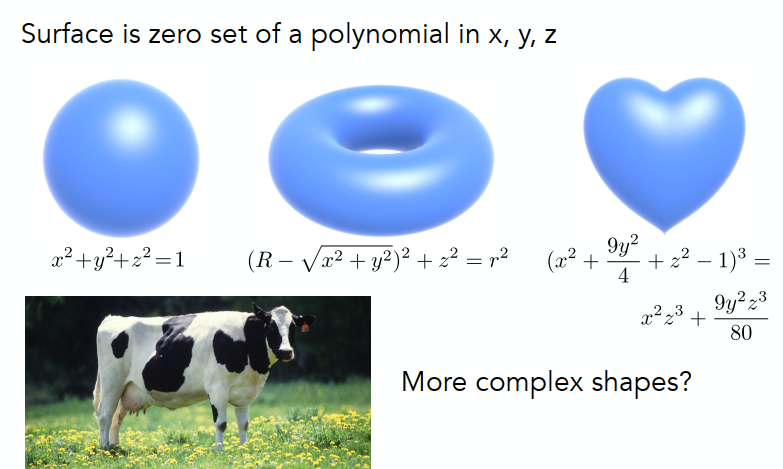
但是显式几何对于判断一个点与几何的相对位置（在几何内还是在几何外还是在几何上）就很困难，通过式子无法得到。

总结：Best Representation Depends on the Task!

# (三)Implicit Representations in Computer Graphics

## 1、Algebraic Surfaces (Implicit)

Surface is zero set of a polynomial in x, y, z



用隐式去表示一些几何当然可以，但是如果只给出式子，不给出几何图形的样子，我们很难通过式子看到几何的“真容”，而且对于更加复杂的几何（如奶牛），如何去做？无从下手！

## 2、Constructive Solid Geometry (Implicit)

Combine implicit geometry via Boolean operations

通过一系列基本几何的基本运算，来定义一些新的几何。

在建模软件中运用广泛（布尔运算）。



## 3、Distance Functions (Implicit)

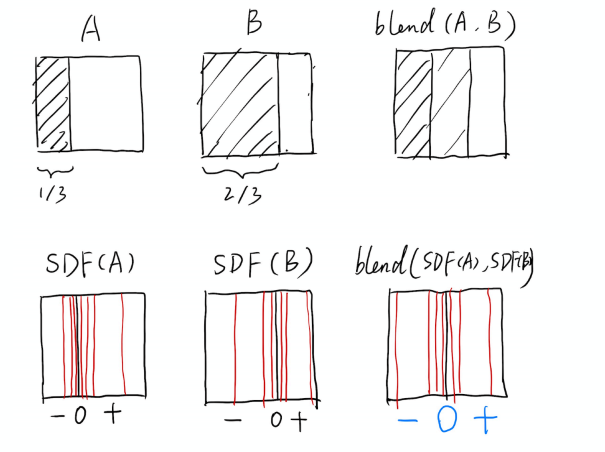
Instead of Booleans, gradually blend surfaces together using Distance functions:

giving minimum distance (could be signed distance) from anywhere to object



距离函数：空间中任意一点到你想要表述的几何形体上任意一点，他们之间的最小距离。这个距离有正负。如果有一个点在几何形体外面，则最小距离算出来之后加个正号，如果在内部最小距离加负号。把这两个物体各自的距离函数都算出来之后，把两个距离函数做一个融合（blend）再把他恢复成原来的物体，就可以得到上图的效果。

An Example: Blending (linear interp.) a moving boundary



如上图，A表示距离左侧1/3面积都是黑色，B表示距离左侧2/3面积都是黑色，A和B做一次blend，得到的结果就是1/3黑色（左侧），1/3灰色（中间），1/3白色（右侧）。

黑 界 白 白 白

黑 黑 界 白 白

Blend

黑 界 灰 界 白

对A单独做一次SDF（SDF（A）），那么就可以得到A上任意一点的距离函数。我们认为A的黑白分界线为0，若一个点越接近于黑白分界线，距离函数的值越小，越接近于0，向右（白）为正，向左（黑）为负。

同理对B也单独做一次SDF（SDF（B））。同样的黑白分界线为0，越接近于黑白分界线，距离函数的值越小，越接近于0，向右（白）为正，向左（黑）为负。

将SDF（A）和SDF（B）做一次blend，得到blend（ SDF（A），SDF（B））。那么这个blend后的图像中间即为0，向右（白）为正向左（黑）为负。把这个blend（ SDF（A），SDF（B））通过SDF再恢复成原来的形状，就可以知道，0的地方就是他们的边界，非0的地方不是。也就是说，blend两个对应的SDF，实际就是在blend他们的边界。

-10 0 5 10 20

-20 -10 -5 0 10

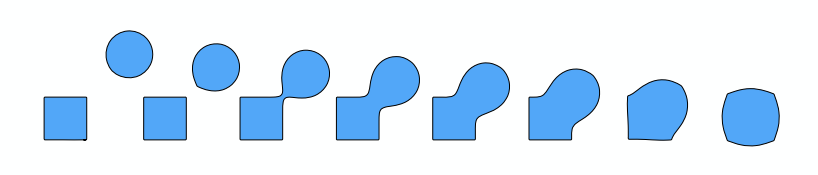
Blend后

-30 -10 0 10 30

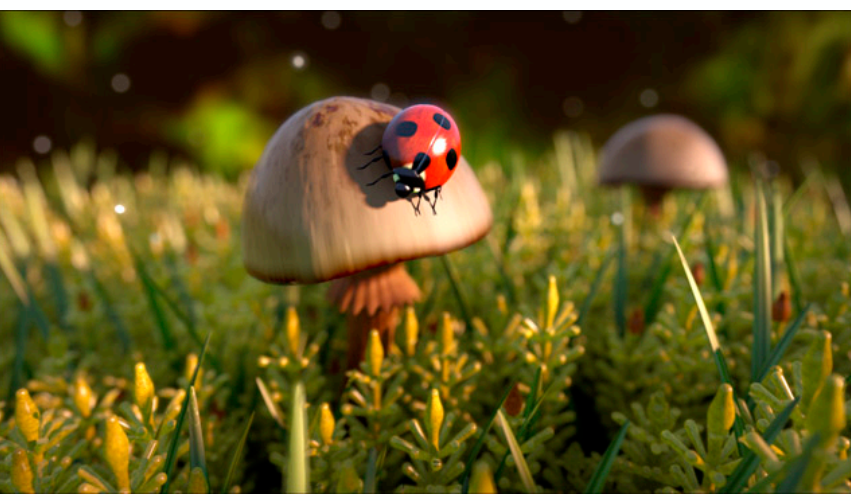
恢复

黑 黑 界 白 白

Can blend any two distance functions d1, d2:



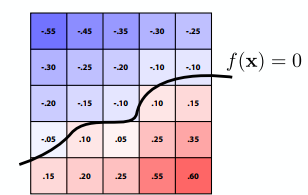
Scene of Pure Distance Functions



隐式几何之间，如果要实现比较圆滑的过度，就是通过距离函数来实现的。

### Level Set Methods(水平集法) (Also implicit)

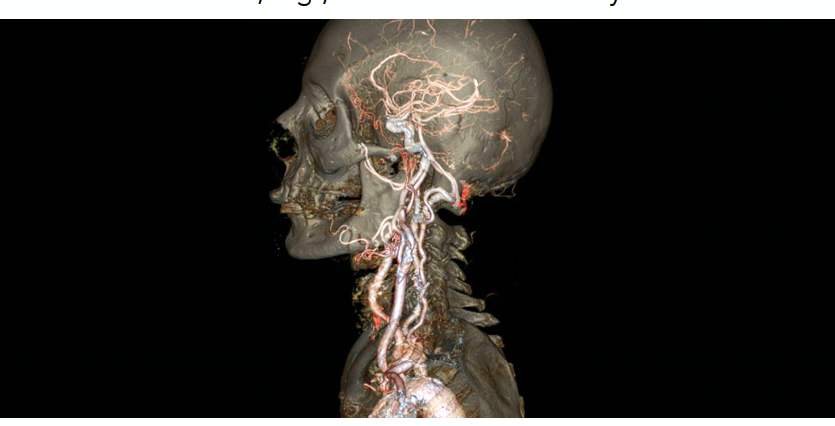
距离函数blend出来之后得到的函数，如何再把它恢复成表面？我们只需要把距离函数对应的0的位置全部找出来（ f（x）=0 ）。



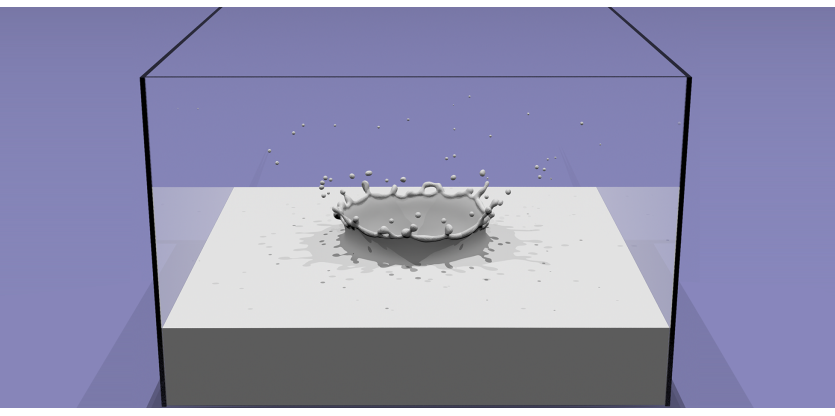
但是当距离函数不太好通过式子表示出来时，可以通过其他方法表示出来，如上图，通过水平集的方法表示。把函数的表述写在了格子上，只需要在格子上找到值为0的那条线，即为表面。（和等高线类似，在不同位置有相同值）至于在什么地方应该等于0？前面已经学过双线性插值了，可以解决这个问题。

### Level Sets from Medical Data (CT, MRI, etc.)

距离函数除了定义在上图的二维格子中，也可定义在三维格子中。如医学成像



### Level Sets in Physical Simulation



包括这种水花四溅的效果，水花和水花融合在一起，也可以通过水平集（距离函数）实现。

## 4、Fractals (分形)(Implicit)

分形就是自己的部分和整体长得非常相似。如著名的科赫雪花：每个边上都有六边形，在看更小的地方又有六边形，再看更小的地方又有一个个六边形（递归）。



中间这个植物（西蓝花的一种），看这个植物，有很多凸起，看一个凸起上又有很多凸起，再去看一个小的凸起又有很多凸起…

分形在渲染的时候会引起强烈的走样，因为物体变化的频率实在是太高。