计算机图形学和虚拟环境读书报告

本文为了将两本教材的相关内容进行整合，没有完全按照课本顺序编排，在每一个小节的标题后附上了两本书对应的章节。

一、概论

I. 计算机图形学的历史【黑书1、白书1、2】

计算机图形学是一个年轻却又充满生命力的学科。其历史可以上溯到一个经典的画板系统Sketchpad，一个诞生于1963年的交互系统。人们利用光笔在屏幕上画图，这标志着图形对象的创立，将计算机从图片中分离出来。

1962年，第一部游戏星球大战诞生了；1973年，第一部具有GUI的操作系统Alto诞生。这些都预示着一门新的计算机学科即将登上历史舞台，而当80年代个人电脑开始走进千家万户、处理能力得到巨量提高的时候，图形学的花朵终于绽放了。

第一个图形学标准GKS在1985年诞生，但是这一标准是基于2D的。很快，1988年PHIGS将这一标准推广到3D去，并称为了新的标准。1992年，OpenGL诞生，它就如汇编之于计算机底层一般，在图形学中占有非常重要的地位，并且沿袭至今。

图形学如今有着非常广泛的应用场景，而大体而言可以分成三个方向：渲染、几何和模拟。其中居于核心地位的是渲染，也就是如何在屏幕上呈现一个虚幻世界。

在计算机的世界中，当我们通过程序和代码构造一个虚幻的世界，呈现在光栅的界面之上，这样的世界就变成了虚拟环境。虚拟环境有三个部分组成：内容、几何和动态特性。在这样的世界中，世界由一个个物体构成，这些物体各有自己的界限，而物体之间可以互相发生干涉。人类位于世界之外，接受着这一虚假世界之中的信息。

有许多设备都可以实现这样的真实感。比如各式各样的光栅化显示器，虚拟现实的头盔，定位的数据手套，还有CAVE.不论是过去的设备还是新的设备，对于人类而言，最主要的一点就是，这个世界真实吗？

在计算机图形学中，真实是一个很复杂的概念。从几何的角度上来讲，几何上的真实是和实际物体的相似，比如一张人脸，由于计算机的建模一般由一系列的三角形构成，如果三角形数量过少，就会导致渲染结果棱角分明，缺乏一种真实感。从光照的角度来讲，“一日之内，一峰之间，气候不齐”，不同位置的光照结果会造成很多不同的效果。光照的计算是非常复杂的，又可以分成全局光照和局部光照等方面。从行为的角度来讲，人想要和虚拟物产生共鸣，就需要某些行为契机。这就是为什么动画中关注于一些动画角色的动作和行为是否与人类相符合，

这里总会引入美学层面的争端：到底何为真实？现实主义认为，“艺术模仿自然”，艺术为表现现实而服务。唯美主义则提出“自然模仿艺术”，追求一种形而上的美的绝对标准。近代，印象主义学派的发展则给图形学提供了更多的借鉴：与其关注显示事物本身，不如关注事物本身的某些抽象特质，就像海德格尔所描述的梵高的那双鞋：在鞋中所看到的世界，是一位迫于生计的饱经风霜的农民。在人类的意识世界中，事物之为事物是由某些标志特征构成的。可是现在，我们不妨抛弃这些外在接收到的信息，关注于模型本身。

真实感一般意味着一定的帧率，需要足够的渲染效率。而反过来，由于算力限制，这周实时性并不能很容易达到。尤其在VR中，当我们进行运动的时候，经常会发生场景明显的阻滞，这就是无法做到实时所带来的真实感的危害。图形学总是在做妥协——在真实的渲染和效率的牺牲之中妥协。人又是贪婪的，总希望能用更少的时间渲染出更真实的结果。正是这份贪婪，结出了渲染领域的果实。如今，实时级别的Path tracing已经投入工业界之中，在未来将会有更加广泛的应用。

回到虚拟世界中，我们往往需要一种沉浸感。我们明知眼前的世界是虚假的，但还是不由自主的感到一种置身其间的感受，这正是虚拟环境的魅力所在。

实现这种虚拟环境，往往依赖于人类的视觉特性。人的视觉本身就是虚拟的，因为能够感知光的只有一小片区域，并且有一系列的滤波机制。人的心智模型会对感知造成影响，并且对信息进行筛选、处理和完善。如果推广到三维，人总是得到一种透视机制。如果想要看到连续的场景，需要借助于视觉暂留现象。总之，这一系列的沉浸感都建构于人的视觉特征之上。

因此，整个图形学的脉络就已经搭建完成了：如何建构一个虚拟世界出来？首先需要数学的形式表达。接下来，要表达出空间中的物体，就需要各式各样的曲线和曲面，并讨论其位置关系；进而引入渲染方程，描述物体表面性质和光线作用情况；最后我们探讨一些计算机动画和模拟仿真。

II. 数学【黑书2】

数学是图形学搭建的基础。从线性代数到微积分，以至于概率论相关的东西，都构成了图形学的基础。书中集中介绍的数学知识，是对线性代数的一个推广。

在3D数学中，三维坐标可以代表一个点，也可以代表两点间的差值。在此基础上，定义了一系列向量的运算：向量的加法、减法、模、点乘、叉乘。这些运算共同构成了矢量代数的基本，也是图形学大厦的地基。

对于向量方向的描述，我们一般采用规范化的做法，转换成三个方向余弦。但是在方向的表示上，直角坐标系有不直观之处，因此球坐标也是方向表示的重要手段。在球面上迫切的需要一个类似于角度的量度，这就是立体角。

图形学中，变换是一切运算的核心，最常见的变换就是仿射变换。仿射变换是一种具有线性性的变换，但遗憾的是无法用最常见的矩阵来表示平移变换。解决办法就是将三维向量推广到齐次坐标，引入w-coord，来实现一系列变换的复合。

现在，我们已经可以表示平移、伸缩、旋转、错切、镜像等常见变换了。将这些变换合成起来，实际上只需要将变换矩阵做乘积；而根据其几何意义，逆变换也可以方便的求出。唯一的麻烦就是旋转变换，绕坐标轴的旋转当然可以直接用矩阵形式给出，但是如果旋转是绕任意轴的，罗德里格斯旋转公式又显得过于臃肿和繁复。于是，四元数登入了图形学的舞台。

四元数本质是对一般复数的推广，所以可以进行一切复数有的特性，比如加法、数乘、模、逆、共轭。但四元数可以进行叉积，这种叉积运算的性质保证了四元数在旋转上的可操作性。同时，四元数还可以进行插值、样条等运算，实现非常平滑的效果。

III. 渲染方程【黑书3、15、白书11】

为了引入渲染方程，首先需要从物理层面上解释光线的传播。光是一种辐射能量的聚合体，假设光具有独立的波长、稳定的能量分布和真空中光运动的媒介。在单位面积上某个方向的通量记作Luminance。

我们可以用光通量来描述单位时间流过表面的能量，也就是Radiosity。在这个基础上对立体角求微分，得到Radiance。换言之，Radiance就是单位时间单位面积上的光通量。

假如两个面片之间有一定的倾斜角度，就需要引入投影连线和法线的夹角。这样，可以定义一个面片之间的形状因子，这个形状因子只和夹角与距离有关，是一个定值。

现在考虑某个面片的光通量，是由自身发出去的通量和其它所有面片乘上形状因子与反射率构成。这样，我们可以将这种形式转换为一个矩阵方程，称为辐射度方程。

在物体表面上，最核心的讨论话题就是反射多少，如何反射。一个极端情况是漫反射，入射光线在各个角度均匀的发生散射；另一个极端是高光反射，以类似镜面的方式将入射光进行反射。用来描述这种分布的就是BRDF函数。

简而言之，BRDF是对确定入射光线下不同出射光线的辐射度的描述。引入BRDF之后，就可以引入Rendering Equation。Rendering Equation由两项构成：前者是自身发出的光，后者是总的反射光。反射光是对所有入射光的累加，所以就需要在球面上对BRDF和入射Radiance之积乘上Cosine值对每个立体角累计求和，也就是积分。

解Rendering Equation的过程贯穿整个图形学的核心。Rasterization方法是最古老的解决策略，同时也是最高效的一种办法。但是这样得到的效果只能说差强人意，逐渐无法满足人们日益增长的对真实性的渲染的需求。Whitted-Style Ray Tracing能做到真实的渲染，但是由于其过于昂贵的算力开销，一般只能用在离线渲染中。针对其过高的复杂性，Path Tracing得到了广泛应用，但由于Mont Carlo积分的采样操作，仍然不能做到实时级别。直到最近，实时Path tracing终于在复杂度和真实性中间达成了良好的妥协，将会得到越来越广泛的应用。

IV. 颜色【黑书4】

颜色是虚拟和真实的分界的一种象征，历来被哲学家所探讨。一个有趣的悖论是色盲悖论：如果一个人与生俱来就是色盲，那么即便他所认知的社会和其他人不同，他所认知的也是自己所看到的世界。从这里引发的认识论之争，并不在讨论范围之内；但是我们总是习惯于从此引发思考：看到是真实世界，真的是真实的吗？

从光本身的角度来说，光在不同波长有不同的密度，形成一种分布函数。这样的光谱分布在不同细胞的过滤效应之下，形成了人的认识。所以人对颜色的观察总具有一种积分的形式；也因此，我们可以引入Delta函数来描述单色光。

人有三种视锥细胞，即L型、M型和S型。三种细胞有各自的反应函数，对不同波长下光分布积分形成的三元实数对(l,m,s)就是对一个颜色的三元实数对描述。如果用三个固定的阶跃函数替换自然界中的某些分布，就得到了一个由三个颜色反应基函数构成的线性空间。任意一种颜色都可以在这一空间中找到，同样可以得到(l,m,s)的三元实数对。

接下来可以引入颜色匹配函数。根据这种线性性，我们试图找到每种基色的强度权重，通过加权获取目标颜色的匹配。最常用的匹配就是CIE-RGB系统，得到的权重就是三个参数(r, g, b)。把这种颜色进行投影，得到了其色度空间，也定义了其亮度。

但是自然的RGB空间存在负数区域，不易于表示，XYZ空间很好的解决了这个问题。XYZ空间的X和Z是零亮度，从而获得了很完美的CIE-XYZ色度。

在一个CRT显示器上，每个像素点由R、G、B三种荧光色点构成，组合成一个显示颜色。荧光点刷新频率很高，可以实现60fps的刷新率。不同的像素数目可以用分辨率，来描述，如果分辨率不够高，把像素看成点会形成走样现象。内存中有一个帧缓存区，可以刷新到屏幕中间。

由于RGB值是控制光束电压值的描述，而电压和发光强度并非线性关系，所以在处理的时候还需要加入Gamma校正，一般取Gamma为2.2。

当我们了解了颜色空间之后，就可以真正的考察场景是如何构造出来的。

二、光栅化

光栅化是最早被应用的渲染技术，也毫无疑问是实时渲染的核心，无论什么时候都必然会被广泛应用。光栅化的实际上是从相机出发，通过对视锥体中的物体的裁剪和变化，投影到屏幕之上，然后离散化成一系列像素，经过着色、纹理映射等过程，最终呈现出来的的一个流水线。把握光栅化技术，核心就是对渲染流水线的把握。

渲染流水线大体可以分成三个阶段。第一个阶段是CPU的系统调用阶段，向GPU发出渲染指令，这个过程叫做Draw Call。第二个阶段是几何处理阶段，进行模型和观察变换，之后对顶点进行着色，再经过投影和裁剪，最终映射到屏幕上。这个时候，屏幕上是一系列三角形顶点，光栅化阶段就是把这些顶点转成像素。首先要设置三角形，然后进行三角形遍历，对片元着色器进行着色，最终把不同的颜色进行混合，就得到了最后的结果。

I. 相机模型【黑书7、23】

在图形学中，常见的相机模型有两类。一类是针孔相机模型，一类是透镜相机模型。在光栅化中，一般直接使用的是前者。但如果希望加入景深的屏幕后处理效果，可以使用后者。

对于一个相机，我们使用三个向量来进行描述：Position，Up和 Look at。三个自由度将相机完全确定：Position决定相机的位置，Look at决定相机的观察方向，Up决定相机向上的方向，这就是观察坐标系，也称作UVN系统。

针孔相机是最简单的相机模型，类似于小孔成像，在孔后面放一个传感器，就可以取得成像结果。对于这样的摄像机，看到的是一个椎体区域，称为视锥体。视锥体的张角记为Field of View，也就是FOV。

而对于使用透镜的照相机，可以用参数方程来描述光线的传播过程。这样，在焦点处，光线汇聚效果最强，离焦点越远，就越会形成一个光圈，实现模糊。如果引入一个参量来描述能够清晰成像的范围，即Depth of field，就可以找到模糊的范围。这种模糊也可以推广到运动模糊上，通过子像素分割来起到滤波的效果。

Light Field是一个研究方向的热点。如果用任意表面的p和所有方向w这样的五维变量，构成的域就叫做L域。在场景凸包外侧，可以找到一些光片，得到一个有关光域光亮度的连续函数，从而构造出一个网格。

II. 投影【黑书9、白书4】

在投影过程中，有几个非常重要的参数。视平面距离是从相机投影点按照Look at方向的距离，在这个距离上，由近平面和远平面中间包夹的区域形成主要投影区域。此外，视口的长宽比也对裁剪起到很大作用。这些参数共同构成的一个空间区域称为视锥体。

裁剪有两种类型，一类是平行投影，一类是透视投影。平行投影的视锥体是一个长方体，物体从场景中点开始做平行线，与投影面相交，得到的图形就是平行投影。因此平行投影无法显示远近关系，某种意义相当于无穷远点上的投影。透视投影将所有直线聚合在进出的某个点处，形成的叫做等角投影。这种投影能表现出来大小和远近关系，具有良好的性质，也更接近现实世界中看到的东西。

为了计算投影，首先需要建立正则化的框架。平行投影的正则化框架是一个长方形，而透视投影则是一对相交直线。根据相似三角形，可以分别将这些点转换到正则化框架中。对于平行投影，只需要将所有的点进行放缩，直至都落在[-1,1]之中；而对于透视投影，将点和观察点进行联系，可以找到相似三角形，进而得到新的坐标。

这一过程可以通过一个矩阵来进行，综合得到的就是透视投影矩阵。这样得到的齐次坐标的w分量可能不是1，所以通过除法就可以得到所有点的最终投影。

接下来，我们可以把z分量舍弃掉，然后经过视口变换，这样就得到了最终图形在每一个像素点上的颜色信息，这就是整个光栅化的过程。

在此基础上，可以建立立体视图，通过对左右眼图像进行分别构造，实现3D图像。如果我们知道双眼瞳孔间距IPD，那么对于一个点，依据在图像平面的左侧还是右侧，就会得到不同的投影点。这样，看上去不在一起的点却在视野中形成了一个新的虚点，完成了图像平面到世界坐标的映射。

立体重叠是一种很重要的思想，两只眼睛看到一张视平面窗口，会在视锥体上形成重叠的公共区域，如果重叠区域很大，那么最终的成像会比较有立体感；反过来，如果瞳距很宽，就会影响成像结果的深度。

III. 裁剪【黑书10、白书4】

在得到投影结果之后，接踵而至的就是裁剪，我们需要确定究竟什么是可见的，而什么是不可见的。有许多不同的算法适用于不同的裁剪过程。

Sutherland-Cohen算法可以用来裁剪直线。这一算法分成两个步骤，一是判断直线段是否在窗口内，或者完全在窗口外。如果都不满足，就需要首先求出和窗口边界的交点。这样，线段就分成了两部分，一部分完全不可见，一部分无法确定。按照这个过程进行递归，就实现了直线裁剪。也可以使用二分的思想，从中点开始逐个分割，直到在给定精度内，判断所有细分的线段是否可见。

Sutherland-Hodgman算法是最为基础的裁剪算法。这个算法思想很简单，首先构造多边形的顶点序列，然后讨论边和裁剪平面之间的关系。如果线段进入裁剪区域，那么这个点进入栈中；如果离开就送入栈中，并发生匹配，执行裁剪。但这个算法可能导致重叠现象，所以Weiler和Atherton对此提出了改进。

WA算法的核心是对边进行讨论，因为裁剪结果中的边要么是需要裁剪的多边形中的边，要么是裁剪区域边界中的边。因此，按照某种顺序遍历区域，建立所有交点，然后建立一个环形链表，表中存储所有的顶点。对这个环形链表执行遍历操作，就得到了分立的一系列集合，也就是裁剪结果的集合。

在三维空间中，SH算法很容易进行推广。在投影空间中，可以计算出方程与直线相交的结果。我们将直线推广为平面，这样SH算法可以通过直线穿出还是进入平面计算出来。但是投影空间中的裁剪可能出现问题。因此可以考虑在规范空间中借助不等式裁剪。

也可以在齐次坐标中进行裁剪，这样只需要比较x和w的大小关系，如果变换后x在-w到w的范围内，就说明不需要裁剪，同理对y也有这样的关系。

IV. 可见性确定【黑书11、白书8】

可见性问题是一个非常重要的问题。比如，有两个三角形相互重叠，那么究竟看到哪个三角形取决于三角形的位置关系。这种处理大致有三种方法，一种是在对象之间比较，然后连续的求解覆盖区域，虽然能求到精确解，但是鲁棒性不好，实现很复杂；一种是直接对图像进行操作，通过对每个像素的位置关系进行比较，得到最终比较结果；最后一种是上述两种方法的混合。

可见性首先需要考虑背面剔除的问题。物体的背面是一组法线与摄像机相背的面，通过平面方程ax+by+cz+d=0的建立可以得到法向量(a,b,c)，与摄像机的Look At方向做点积的符号，就可以判断出来是在正面还是背面。另一种方法是在投影空间中，只需要判断齐次坐标中z分量的正负，如果是负值，那么就是在后面。

传统的画家算法是从前到后渲染的。对一系列多边形，按照深度从深到浅进行渲染，先渲染第一个，再渲染第二个，这样依次进行。但是如果使用这种算法，就会出现多个三角形之间相互重叠，而无法确定到底谁在前、谁在后。

Z-buffer是最为流行的可见性算法。它的原理基于一个帧缓存关联数组，通过对深度信息的维护和更新，实现对可见性的判断。这一算法非常简单，只需要比较新的像素的深度和数组中维护的深度，如果新的更浅，就渲染并将其更新。

利用画家算法的最简单方法就是直接在投影空间中进行排序，即对进行投影变换后的齐次坐标的z值进行排序。这里要进行排序需要同时判断P、Q的max、min值，来保证二者的相互位置关系是否在同一区域上投影发生重叠。将这种思想得以运用的就是Newell的深度排序算法。

但是这种算法不太精确，对于任意多边形，首先可以定义什么是前后。假如从摄像机出发，能够找到一条光线和两个多边形都相交，那么称二者有深度关系；否则二者可以按照任意顺序渲染。这样，我们可以建立分割平面，即将二者分为两组对象的平面。这样，我们可以递归的建立一个树形结构，即Schumacker分割树，将所有的对象进行划分。其难点在于分割平面的构建，所以很多时候不得不依赖于手工实现。

BSP算法是对Schumacker分割树的一个补充。如果多边形的法向量与某个点的方向向量之点积大于0，那么就说点在正面。BSP树算法分成两步：树的递归建立和遍历渲染。

首先选取一个平面，将场景上的所有多边形分成平面的前、后和平面上三种。接下来按照类似的方式进行递归，平面就叫做分割平面。一般可以使用多边形平面作为分割平面，来简化可见性操作。这里的难点在于，如果平面和多边形相交，就需要把这个多边形分成两个，然后再进一步进行递归。

接下来进行渲染操作。根据Schumacker分割树的思想，和视点在同一平面侧的点不会受到遮挡，所以与LookAt向量在异侧的应该先进行渲染。这样遍历BSP树，复杂度只需要O(n)。由于计算树的代价很大，但是照相机的运动不会改变树的性质，所以这种树适用于静态场景、但动态变换视角。

同时，BSP树能解决很多问题，比如碰撞检测和可见性。建立BSP树的难点是对规模的控制和形状的选择，一种常见的选择方案是选取几个候选多边形，然后对子空间的规模和分布进行评估，选取其中最优值作为根节点。我们不需要关注树是否平衡，但是需要防止拆开过多的多边形。

从树中删除一个多边形比较复杂，需要依据多边形的位置而定。如果多边形在叶结点上，可以直接删除；如果和其它面共同在一个节点上，也可以直接删除，但删除之后需要注意是否需要因为面朝向改变而交换子树；如果这个节点只有一个非空子节点，那么将其直接后继进行代替；如果有两个非空子节点，会得到两个不相干的子树，那么从较大的树开始向下重新建立BST树，相当于进行了重构。

V. 图形的光栅化【白书3、7、黑书12、13、17】

先来看最简单的图形——直线。最容易的方法是使用DDA方法，根据微分方程建立一个递推公式。但是这种朴素算法需要用到浮点数，效率比较低，Bresenham则是对其的改进，也是最为常用的直线生成算法。首先建立y(i+1)=y(i)+m(x(i+1)-x(i))，然后得到y(i+1)=y(i)+m.在此基础上引入误差函数e(x(i+1))=y(i+1)-y(i)-0.5，然后建立递推公式，e(x2)-m-0.5。这样，我们完全回避了除法。也可以在此基础上以步长为2绘制直线，效率得到了进一步提升。

接下来看圆弧的Bresenham算法，考虑到圆的对称性，对于(x,y)有7个旋转对称点和轴对称点。从(x,y)出发判断弧走向，建立d(i)=(x^2h(i)+y^2h(i)-R^2)+(x^2l(i)+y^2l(i)-R^2)，由于递减性质，可以根据正负判断下一个节点在上方还是下方。那么d(i+1)=2x(i)^2+4x(i)+2y(i)^2-2y(i)-2R^2+3，这就是其递推公式。也可以通过多边形迫近圆，然后用直线生成算法，来绘制圆弧。

之后我们就要考虑对于多边形的着色。多边形有两种表示方法，一种是顶点表示，也就是通过顶点序列来维护多边形；一种是点阵表示，通过内部像素集合来刻画多边形。前者适用于内存中的表示，而想要在图形中实现渲染，需要将其转换为点阵表示。这个过程叫做扫描转换。

最简单判断一个点是否在多边形内，可以使用累计交点。如果从一点出发的射线和多边形交点个数是奇数，那么点在多边形内，否则在多边形外，而如果遍历每一个像素，就可以得到每个像素的内外关系。这一算法复杂度非常之大，不可取。所以我们有必要引入扫描线算法。

扫描线算法实际上是利用了像素之间性质上的连续性，这种连续性表现在三个层面，一是区域连续性，也就是对一个长方形区域，和多边形相交得到多个梯形，如果能确定梯形中一个点的内外关系，那么整个梯形的内外关系也随之确定；一个是扫描线连续性，一条线和多边形相交得到的扫描线基于奇偶性分割为内部和外部的顶点集合；一个是边的连续性，不同扫描线上点的坐标查是其斜率的倒数。

这里还有一个边界条件需要处理，就是扫描线和图形顶点相交时候的情况，称为奇点。这个时候为了保证奇偶性，需要依据奇点的类型，将奇点分成一个或两个点：如果是极值点，那么就当作两个点；否则当作一个点。

接下来建立两个链表。对于一个边，维护上端点的y坐标、下端点的x坐标、斜率倒数和指向下一个边的指针，依据下端点的y坐标分类，再对x排序，建立分类表ET；将当前扫描线相交的所有多边形边组成边的活化表AEL，并根据边的连贯性刷新交点序列。

可以按照如下的过程执行扫描线算法：先取出ET表中的最小元素，然后从下到上执行扫描线，直到ET和AEL都是空为止。再这个过程中，首先将此时的ET表中边取出，然后按照x从小到大排序插入AEL。如果AEL非空，说明此时扫描线构成了一系列在多边形内的交点，这个时候以两个为1对进行匹配，然后着色。接下来删去AEL中扫过的边，然后让边的x域进行累加。

上面的扫描线算法需要用到排序，求余是对其的一种优化。假设A表示对像素着色，I(A)表示对某种操作取反，那么I(I(A))=A。现在，假如扫描线和一系列的顶点序列A1,A2,...,An相交，那么我们首先把从A1开始把右侧的所有像素都取反，接下来A2、A3……，这样就得到了正确的着色结果。

借鉴这一思想，如果以边为单位，把每个边向右的部分都进行着色或逆着色操作，这样也能得到正确的着色结果。这种扫描转换的方法就叫做边缘填充算法。但是这中间会需要对变量进行大量着色操作，可能降低效率。可以把扫描线和边缘填充算法，首先光栅化得到所有的边界，并标志为特殊像素，然后调用扫描线算法，把中间的部分填充颜色，这种方法就是边界标志算法。

除去扫描转换，区域填充也是一个重要问题。区域填充是把整块区域都换成某种颜色，类似于floodfill。这种区域一般分成两种，一种是通过区域内的所有点表示，一种是通过区域表示。可以使用简单的DFS进行填充，但是这种方法效率比较低；也可以通过扫描线的方法，先向左右扩展，然后再移动扫描线，找未填充颜色的区域。

VI. 反走样技术【白书7】

只要谈光栅化，就往往无法脱离走样——因为光栅化意味着在固定的某些点上着色，而产生一些不真实的效果。因此，有必要引入反走样，来实现更加真实的渲染效果。

这种采样一般是由于采样率不足造成的。如果把渲染的过程比作一个信号处理，那么在这个过程中，如果实际的变化程度很快，而采样率不足，信号发生重叠，在显示上就会出现摩尔纹等效果。因此，可以考虑在时域上的卷积操作，即频域上的滤波操作，某种意义上就是加权平均。

对于线段，可以假设线段有一个粗细，然后通过这个面积和某个像素的交来作为灰度值，这样线段的颜色就会变得平滑了。对于多边形，可以通过求解多边形在某个像素上交的面积作为灰度值，当然这些做法都比较复杂。

把这种方法推广到一般的几何体上，假如某个像素中图形占据了一定面积，就可以按照这个面积进行灰度值的赋值。从信号处理的角度来讲，这种处理办法实际上就是拿一个1x1的低通滤波器进行滤波。

由于计算重叠面积是一件非常复杂的事情，所以工业界常使用MSAA进行近似。MSAA的核心是进行过采样，即对原本只需要采样一次的像素进行多次采样，然后将结果进行平均，以得到卷积结果。这种算法只需要在渲染循环中的采样数多3倍，然后将测试结果当作覆盖面积，转换成灰度值。

VII. 着色【黑书6、13】

接下来需要考察的问题是物体表面如何和光线进行作用。物体的表面性质是具有复杂性的，这种表现在表面形状是多样的，同时和光线的作用模式也是多样的。一个平滑的表面具有比较均匀的法线，而某些粗糙的表面可能法线分布非常混乱。

在这里，首先需要提出三种最典型的材质。第一种是Diffuse的材质，这种材质在光线作用的过程中，反射结果会均匀的向四面八方散射。从BRDF的角度来看，就是这种表面的BRDF函数是一个常数。第二种是Specular的材质，这种材质的BRDF函数接近于一种阶跃函数，会在反射方向上非常尖锐。第三种是Glossy的材质，介于二者之间，BRDF分布不是特别均匀。

对于不同模型，最为经典也最为流行的就是Blinn-Phong模型。这个模型将光线对物体的作用分成了三部分：ambient，表示环境的部分，这部分在早期渲染中一般使用常数作为近似，现在也可以用某些算法得到GI的贡献来作为环境项；diffuse，表示漫反射部分，由于光线和法线夹角越大，表明在表面上接收到的光线越少，所以可以使用光线和法线夹角的cosine值来表示这种非线性关系，然后乘上漫反射系数和光强，得到出射光线结果；specular，表示高光反射部分，这部分主要关注反射光和观察视线的夹角，因为如果反射光和观察视线很接近，就说明这个方向上接收到的反射光比较多。将这个点乘结果进行幂运算，就可以操控高光范围，将这个幂称为高光反射系数，这个系数越大，得到的高光项就越尖锐。

很容易发现，实际上控制某些参数，就决定了某个表面的性质，也就是和光线作用的结果。可以把这些参数抽象为一个结构，就叫做Material。

由于对渲染真实感的追求，现在越来越普及一种基于物理的渲染（PBR），核心就是讨论表面与光线作用的问题。在这种模型中，一般考虑表面对光线的反射作用和折射作用两个层面。有一个经典的定律Fresnel Term，可以给出不同角度下的反射和折射的比重，通过这种方法引入辐射度量学，就可以得到具有物理性质的渲染效果。

VIII. 纹理映射【黑书13】

物体表面可以有五彩斑斓的颜色，而不仅仅只能是枯燥的玻璃或金属器皿——现实世界中也是如此。我们可以在书上印上一张图，可以在杯子上贴上一张贴画，而类似的，我们也可以在物体表面贴上一张图，也就是texture。

可以把纹理当成一张图片，这张图片被划分成一块块的像素单元，每一块单元就是纹素。在模型中，一般已经蕴含了纹理坐标，通过纹理坐标在图像中采样，就可以得到某一个像素的颜色。

对于空间中的几何体而言，由于几何体大多由很多三角形面片构成，而在模型中往往包含了三角形顶点对应到纹理中的坐标是多少。此时，对于面片上任意一个点，只需要通过重心坐标，就可以直接求出来插值之后的纹理坐标值。但这样又有一个问题：如果我们直接进行插值，那么这个时候插值是在裁剪空间中进行的，明显会出现问题；如果先做逆变换再进行插值，效率会比较低下。

解决这一方法可以使用透视投影矫正。也就是，在经历非仿射变换之后，可以证明对1/z进行插值是能够得到正确的结果的。因此，这是对纹理坐标插值时需要注意的一个细节。

可惜事情总不是这么简单。由于渲染要求的分辨率和纹理的分辨率很难相吻合，就有可能出现两种情况：一个像素对应多个纹素，或者一个纹素对应多个像素。对这种情况，一个最为简单的方式就是进行滤波，也就是取加权平均；但是这种方法并不都有效。

如果纹素太大，也就是纹理较小，可能取样之后得到的是小数，而模型变化快就会产生严重的走样。这个时候最常使用的方法是双线性插值：取周围的四个顶点，分别做两次线性插值，来当作对这个顶点坐标的近似。这样相当于还是进行了一个平均。

如果纹素太小，也就是纹理较大，一个像素对应多个纹素，这个时候就需要得到这些纹素的加权平均。如果使用传统的MSAA等反走样方法，代价是非常高昂的，因此MipMap方法得以引入：建立以2的次方衰减的一系列纹理，然后在这些纹理中间快速的查询均值。对于某个确定的像素大小，可以在MipMap中查找到最近的两个纹理像素，然后对两个像素的查询结果进行插值；在查询某个MipMap的时候，需要对对应的像素进行双线性插值，快速得到结果。这样，查询的复杂度就降到了很低。

除去一般的diffuse texture，实际上还有很多其它种类的texture。得到广泛应用的一个是环境纹理，一个是凹凸纹理。除此之外，还有比如用于高光遮罩的遮罩纹理、用于卡通渲染的渐变纹理、用于设定光源属性的自发光纹理等。

环境纹理往往是立方体或者球体，规定了环境光的种类。如果把环境当成一个盒子，将盒子展开之后，就得到了六张图，这六张图共同构成一个CubeMap。从这个盒子中进行采样，就可以采样到周围的环境光，用来实现着色。

法线贴图主要是用来生成物体表面的凹凸性。如果把每个法线的坐标值都储存在贴图中，就可以通过这些值来推测法线的方向，进而获取表面的实际高度。贴图中储存的法线值一般使用法线坐标，也就是以法线、切线来构建一个TBN坐标系，并以此为基础求解实际高度。这种方法存储效率高，也易于使用。

IX. 阴影【黑书14、白书11】

有光的地方就会有影子，如果没有很好的阴影，不论模型多么逼真，都会给人一种强烈的虚假感。因此，如何生成真实而又漂亮的阴影历来是一个很重要的话题。

最经典的阴影生成算法是Shadow Mapping，通过一张特殊的纹理来实现。首先从摄像机出发，进行一次渲染，然后记录下每个顶点的深度，也就是zBuffer。把这个深度通过纹理的形式记录下来，形成的就是阴影图。接下来从视点出发渲染对象，然后进行测试，如果当前深度值比观察空间要大，说明这个物体被遮挡了，应该位于阴影中间；如果相等，说明这个物体就是光源出看到的。可以说，这种渲染方法大体而言是基于图像空间的。但是Shadow Mapping有一个很重要的问题，就是可能出现自遮挡现象。如果两个点对应在Shadow Map上的采样点相同，但二者的实际距离却不一样，就会导致屏幕上出现各种奇怪的纹路。对这个问题的一种解决方案是引入一个Bias项，在距离之差小于某个值的时候再进行阴影判定。但是这个时候又有可能导致某些挨近地面的位置无法被算作阴影之中。

另一种思路是对所有被遮挡的面进行处理。对于一个点光源而言，某个物体遮挡的空间呈现一个锥体，类似于一个金字塔的造型。Bergeron比较系统化的提出了Shadow Volumes方法。假如从一个观察点出发，看向某个像素，这个过程就会和一系列的平面相交。考虑对于一个锥体上的平面，它的法线方向可以和观察方向同向，也可以反向。现在使用一个计数器，如果是反向就让计数器+1，否则-1.这个计数器最后结果是0说明观察的点不在阴影之内，否则在阴影之外的。阴影平面生成是可以预处理的，而相交的判定则使用模板缓冲区，在和zBuffer比较的时候更新正向与背向之差。SV的方法彻底回避了自遮挡问题，但是效率比较低。

自然而然的，可以生成一颗BSP树，即SVBSP树，来描述一个阴影体。这个树描述的是阴影体的一系列表面。由于离着光源近的多边形才有可能投影到比较远的多边形，可以依照相对光源位置进行排序，来划分空间结构。这个BSP中用OUT表示节点存储的是阴影平面，IN表示被遮挡。

上面的方法生成的都是硬阴影。但在生活中，我们常常看到的是一些软阴影，在边缘有一定的模糊效果。这是因为实际上的光源可能不是点光源，而是线光源甚至面光源。因此，有必要模拟来实现软阴影。

软阴影的实现有两种思路，一种是用解析方法求出边界的范围，另一种是通过采样来近似求得边界的灰度。在此之前，首先需要引入Extremal Shadow Boundaries的概念。这个概念是指本影和半影的边界，在这个边界内，阴影是硬阴影，而边界之外的一段距离中，阴影是软阴影。

Nashita和Nakamae提出了如何使用External Shadow Boundaries来构建本影体和半影体。具体来说，本影的平面是光源的点和投射阴影的物体都在后半空间，而半影的平面是光源的点在前半空间，投射阴影的物体在后半空间。找到边界之后，对半影点构造一个类似阴影体的立方体，然后就可以判断一系列点的可见性。

对于采样方法，最为经典的就是PCSS。这个算法是在ShadowMap中，随机选择周围的一系列像素，对每个点进行比较，然后取加权平均值作为采样结果。这种方法依赖于采样的准确性，所以常常使用泊松采样来达到比较好的效果。它的做法可以分成三步。第一步是Blocker Search，计算遮挡物的平均深度，来作为采样面积的一个量度。第二步是计算Filter Size，可以通过相似三角形来求解。最后执行PCF的过程，在某个区域内进行采样。

PCF的效果和效率依赖于采样质量，如果采样数少，那么结果可能会噪声比较多，而如果采样数过多，又会让效率降低。VSSM则进一步进行了假设，即利用Chebychev不等式直接估计某一范围的深度的采样点数，来作为结果的一个近似。其查询可以使用MipMap或SAT来进行优化，得到的是对结果的一个无偏估计。

三、光线追踪

光线追踪可以说是离线渲染的核心方法，对于光线追踪的渲染结果的追求一直是离线渲染的重中之重。最原始的光线追踪算法是由Whitted等人提出来的，一般被叫做Whitted-style Ray Tracing。在此之后，由于辐射度量学的引入，更加真实的Path Tracing得到了广泛的应用，并且在此基础上有BDPT、MLT等方法。另一种思路是使用Photon Mapping，这类方法对Caustic物体会有比较好的效果。

I. 光线和绘画隐喻【黑书5】

在图形学中，我们一般把光线抽象成一条射线。这条射线从一个点出发，并不断延申。假如把人眼类比成一个摄像机，当我们看向场景的时候，许多光线从不同像素出发汇聚到眼睛里。如果说摄像机是一个画家，那么光线就会汇聚到粗糙的画布上的不同位置上。

亚里士多德认为，视觉的产生需要一种媒介，那就是光。我们对各个方向发出一系列的光，当这些光线到达了世界中的某些对象，那么我们就看到了物体。这个概念叫做光线投射，是光线追踪的核心。

假如我们把画布划分成M×N的网格，让(0,0)位于左下角，(M-1,N-1)位于右上角，很容易就可以找出来一种映射，把(x,y)映射到网格之上。每个小网格对应的是一个矩形区域，实际上代表了某一块物体的颜色进行混合的结果。

不妨取整个矩形区域的中点，和人眼的位置二者构成的向量当作光线的矢量，然后投射出去。这个时候光线会和场景中的许多物体相交，求得一系列交点。举一个最简单的例子，就是和一个球心在原点的球面相交，可以很容易把方程解出来。

虽然这种做法耗时比较多，但是它一定是正确的。这个时候，每一条光线都代表了一个前进的方向上的颜色——也就是它所代表的像素的颜色。最后，我们只需要把这个颜色对应的放在窗口上，就完成了一幅画的绘制。眼中所映射出的世界的风景，代表了画上的每一个像素。

海德格尔说，“人诗意的栖居于大地之上”。我想，不仅仅是人，图形学的风景也充满了诗意，因为它在屏幕上呈现出来的风景越是真实，越是贴近自然的本真，就越富有一种让人沉醉其间的吸引力。从这重意义上讲，屏幕上的物和自然之物或许没什么本质区别。柏拉图提出的三重世界中，理念世界充满了意义和价值，现实世界只不过是对这种理念世界的投影，而画家所作的只不过是对现实世界的描摹和临状。可是画本身又何尝不是另一个敞开的世界呢？在这个由画家所描述出来的世界中，人和世界相接触，参与到世界之中。图形学也是一样，在矮矮一方的屏幕上，人和那丰富又充满魅力的世界相接轨，这也是图形学的魅力所在吧。

II. Whitted-Style Ray Tracing【黑书6、白书11】

1980年，Whitted等人将递归式的光线追踪引入了图形学之中。这是一个开创性的工作，因为光线追踪本身得到的是完全正确的结果。

这个过程首先从产生光线开始。从相机出发，向网格化的平面中每个网格发出一系列光线，这些光线都会和物体发生碰撞。那么对于这个物体来说，由于BRDF的特性，光线的作用可能是多种多样的。比如，这个物体是透明材质，光线可以越过表面继续传播，代价是发生一定的偏折；比如，这个物体是富有金属性光泽的物体，光线发生碰撞之后会发生反射，然后继续向其他方向传播。这个时候，这个物体实际上相当于一个新的光源，一般称之为次级光源。从这个次级光源开始传输光线和从初始的光源开始传输光线，实际上是完全一样的子问题，因此完全可以递归化处理。

一般的BRDF可能具有比较复杂的形式，有的时候可以用Lambert模型和Blinn-Phong模型进行近似，也就是Diffuse物体可以继续传输，Specular物体只向某一个方向进行反射。对于透明表面，则可以使用Snell定律，由入射方向和法线方向计算出射方向。一个物体表面可能有不同的传输情况，把这些传输结果加起来就能得到对某个物体的最终着色。

从辐射度的角度来讲，由于发生了反射和折射，光线必然会不断衰减。所以可以假定一个最大弹射次数，就可以达到一个比较好的结果。

III. 光线求交【黑书7、白书11】

想要正确的完成上面的算法，有一项工作是不可避免的，就是光线和物体的求交。在图形学中，这个工作无法绕开的是和多边形的求交，首当其冲的就是和三角形求交。

三角形确定一个平面，这个平面可以用一个点法式来表示。把光线的坐标带入到点法式之中，就能确定光线的传输距离，然后只需要判断交点是否在三角形内。Moller Trumbore则进一步的进行了优化，使用重心坐标描述三角形的方程，然后通过对三元线性方程组的求解找到了一个公式。

对于多边形，由于多边形同样在一个平面内，所以仍然可以求出来一个交点，唯一的麻烦就是判断交点是不是在多边形内，而可以首先把多边形和交点投影到一个坐标面上。一般选择投影到夹角最小的坐标面，这样能保证丢失的信息最少，同时也可以防止积聚等现象的发生。

接下来，按照逆时针方向选择边界点，由于在多边形内的点一定在所有边构成的直线的同侧，所以可以根据某些算法来把所有边的解析式都计算出来，然后带入待确定点的坐标，然后判断最后得到的符号是否相同。

和每个平面都进行计算是复杂度很高的，所以可以考虑用一个盒子来把这个范围进行确定。光线和某个平面相交首先需要和这个包围它的盒子相交，而和盒子求交的复杂性要远低于平面求交。对于一个三维盒子，由于光线需要进出，只需要解和六个面相交得到的t的相应大小，就可以确定究竟是否相交。

IV. 加速结构【白书11、黑书8、16】

细数光线追踪的算法，会发现耗费时间最长的实际上是光线和平面求交的过程。如果直接进行遍历，那么每进行一次光线传输都需要把场景中所有平面都遍历一次，这个开销是不可能接受的。

一种思路是把空间场景进行划分。使用这种结构的包括空间八叉树、KD-Tree和BSP树。八叉树把空间分成等大小的8份，并不断进行递归划分，形成一种树状结构；BSP-Tree在前面已经进行过介绍，这里不多赘述。KD-Tree则依次按照水平或竖直方向进行划分，每个节点先水平的分成两份，然后在两个子节点中竖直的分成两份，就构成了KD-Tree。因此，KD-Tree对空间进行划分的同时，还很好的保持了二叉树性质。

但KD-Tree的应用存在一些问题。一方面，使用KD-Tree就意味着需要把物体切割开，会增加求解的复杂性。另一方面，KD-Tree对场景的划分是均匀的，这对信息分布比较均匀的场景来说无足轻重，但是对物体分布不均匀的场景堪称灭顶之灾。

BVH是当今最常使用的一种加速结构。它实际上是递归的对盒子进行一系列划分，然后递归的对场景进行求交。比如，对于一个三角形网格构成的物体，可以根据最长轴的方向进行划分，划分标准为位于中位数的三角形。这样，场景就构成了一个个嵌套的盒子。这些盒子之间可能出现交叠，但是一个物体出现且仅出现在一个盒子里，最终形成的也是一个树状结构。比起KD-Tree，这种方法容易实现，而且效率也比较好。不过每加入一个新的物体，BVH就需要重新计算，做不到实时更新。

V. Path Tracing【黑书22】

路径追踪实际上是以辐射度量学的基础上发展起来的改进的光线追踪，可以说是现代离线渲染的又一大里程碑。首先考虑渲染方程L(p,w)=Lr(p,w)+∫f(p,wi,wo)L(p,wr)cosθdw。假设把积分符号当成一个L=Lr+RL，那么L=Lr+RLr+R^2Lr+……，也就是考虑多次弹射的光线传输。如果只考虑前几项，这个公式实际上就完成了一个很好的近似。

由于涉及到积分运算，直接求解是比较困难的，所以在图形学中一般使用的是Monto-Carlo积分。对于一个积分，把求解过程转换成采样过程，这是Monto-Carlo方法的精髓。但是这里需要考虑到BRDF的性质，由于对任意一条入射光线，出射光线的方向是随机的。如果使用N条光线采样，那么出射光线将会按照指数级别增长，这个开销是难以接受的。

因此，可以直接选取N=1。由于从视点到像素可以取多条光线，然后取平均，那么最终得到的结果实际上相当于能够遍历大多数情况。同时，常常使用Russian Roulette的方法，对某个表面，按照某个概率打出一条光线，这样就会在期望不变的情况下保证递归终点。

为了缩减方差，也就是去掉某些噪点，Arvo和Kirk引入了重要性采样的方法。具体来说，就是在生成光线的时候不使用均匀分布作为概率分布函数，而是根据具体平面性质来进行采样，得到一个比较小的方差。从这个思路往下发展，得到了MLT等方法。

BDPT，双向路径追踪是另一种优化思路。由于在场景中，可能光源只占很小的一部分，但是从相机出发的光线不容易打到光源上，因为概率比较小。为了解决这个问题，可以同时也从光源出发发射光线，然后让二者发生交汇。这种做法速度较慢，但是对某些场景表现比较好。

VI. Photon Mapping【黑书22】

如果把光子看成一束能量，并且包含碰撞的交点位置和方向，可以建立一颗KD-Tree来描述这些光子，构成了一个光子图。Jensen创建了两个这样的光子图，一个叫做焦散光子图，一个叫做全局光子图。

焦散光子图标识一个路径。这种现象实际上是对一个漫反射表面，当一束光线发生碰撞的时候可能会产生一个特别的图案。而如果光子从光源出发，和表面碰撞的时候按照BRDF进行反射，直到遇到漫反射表面为止，会形成一个焦散光子图。

全局光子图主要解决的是光源的光子分配问题。这个光子图可以沿着任何的路径，密度比较低，用于间接反射。它实际上是Rendering Equation的一个全局解。

在渲染的时候，将Rendering Equation拆解成三部分，分别是光源贡献、漫反射的焦散成分、镜面反射的辐射度。第一项可以用全局光子图的光亮度估计，第二项通过焦散光子图中读到，第三项从路径追踪可以进行求解，这样就完成了Photon Mapping。

与Path Tracing最大的不同之处在于，光子映射给出的是一个有偏估计，但是当光子数足够大，会收敛到一个正确的结果，也就是一个一致估计。

四、几何

CAD一直以来就是图形学的一个重要应用方向，在其中的三维造型实际上是图形学所讨论的一个很大的范畴。

几何大致可以分成两个主要部分。一个部分是场景构造，包括CSG Tree、边界表示法和空间八叉树，另一个部分是参数化曲线曲面的生成，也就是Bezier曲线和Spline。

I. CSG Tree【白书10、黑书18】

如果把物体看作一系列点的集合，那么就可以把集合的运算运用在物体之间的关系上，这就是CSG Tree的起点。这样的运算包括交、并、差等，比如在物体上开一个孔，实际上是一个立方体和一个圆柱体做差运算；比如在圆柱体上放一个立方体，实际上可以看成一个并运算。但这种运算存在一定问题，也就是会出现悬边和悬挂面。因此，引入了正则化的集合运算，先进行运算再求闭包。

那么可以想象到，在构建一个三维物体的时候，可以用这样的一颗树来表示所作的操作。树的叶子节点是构成几何体的基本物体，叫做基本体素；非叶子节点表示集合操作。如果对这颗树进行遍历，就得到了一个物体的构造过程。

这样的数据结构一般由五个元素构成，分别是操作类型、坐标变换、基本体、左子树、右子树。如果建立出一颗CSG树，想要计算某些几何性质，往往就可以递归的求解。比如希望求出物体的重心，就可以按照dfs序统计每个小立方体的重心，然后构造在一起。

另一个问题是光线投射，即显示物体的精确边界。从视点出发，向每一个像素投射光线，就可以判断出最近的可视交点是哪个。对于这个问题的求解，可以先对每一个基本体素求交，然后再对这些交点递归式的运算，最终找到距离视点最近的交点，就得到了光线投射的结果。

CSG Tree简洁有效，很适合表示工程中的基本元素变化，并且表示简洁。但与此同时，这种表示方法无法对物体表面进行修改，进行很多操作效率都比较低。因此，常常使用边界表示法。

II. 边界表示法【白书10、黑书8】

三维物体往往可以用一个可定向二维流形曲面的三维空间集合来表示，这实际上是把一个二维流形同构到了平面图上。具体来说，这种同构是一个映射：图中的每条边对应物体的一条边，每个顶点多边形形成一个循环。

因此，边界表示法就是用边界来表示物体，对于三维物体实际上就是构成外表面的一系列平面。顶点、棱边、表面之间相互连接，一共有9种表示方法。在这种表示方法中，物体与物体的关系信息叫做拓扑信息，物体本身的特性叫做几何信息，二者共同构成物体的信息。

边界表示法常用的数据结构是翼边数据结构，由Baumgart提出。翼边结构是由边构成的结构，它的数据结构包括两个顶点的指针、所在曲线的信息、两个环指针。这两个环分别是指边的左侧和右侧的绕环，这是由于多边形的循环性决定的。同时，在此之中还需要存储左侧循环的第一条边、最后一条边、右侧循环的第一条边、最后一条边的一共4个指针。

在翼边数据结构的基础上，又提出了半边数据结构。这种结构考虑到两个循环的不便于维护性，把一条边拆成了零部分，保证一条边只位于一个循环内。从拓扑结构上，半边数据结构分成体-面-环-半边-顶点五个层次，通过循环来相互沟通。

接下来，需要引入一些造型中最常用的方法，也就是Sweep。Sweep运算用来扫掠，常见的扫掠有平移式的、旋转式的、广义式的三种。平移式扫掠就是平面按照某方向移动一定距离产生新的几何体，旋转式就是平面绕着轴旋转产生立方体，广义式则是将平面区域按照任意轨迹移动。

欧拉运算常被用于数据结构的生成。这种欧拉运算常常有几个比较重要的：mvaf表示输入顶点，开始构造物体；mev表示输入点，连接两个点；mef表示输入边，构造一个邻接面；kemr表示删除桥边、建立新的内环；kfmrh表示删除原有内环，再构造新的内环，相当于开通孔。

除此之外，集合运算也常被使用在这种结构中。另外，可以使用一些局部运算，比如改变几何定义、表面线性变换、物体粘合、某些表面旋转、对称映射、倒角等，来实现复杂的构型。

III. 八叉树表示法【白书10】

在边界表示中，物体的集合运算相当复杂，所以效率很低。因此，用八叉树来做空间划分也是应运而生的一种策略。前面已经引入过BSP-Tree和KD-Tree，再引入八叉树也就一点也不突兀了。实际上，这三种树形结构虽然提及的场景不一样，但是都可以用来做空间划分，完成各项操作。

八叉树将空间分成8份，这种均匀的小格子。然后递归式的细分，构成场景的一颗八叉树。对于每个节点，打一个标记，也就是EMPTY、PARTIAL、FULL。EMPTY和FULL都作为叶子节点，无需继续细分，PARTIAL则继续递归，直到分割出来每个小立方体的边长是单位长度。

建立了八叉树之后，物体的集合运算就转换成了对八叉树的结构变换，只需要对树形结构进行修改，就可以实现并交差。隐藏线和隐藏面的消除也变得十分简单，因为八叉树的元素已经是按照空间顺序排好的，所以可以在线性时间内找到隐藏的线面。

八叉树的主要缺点是占据了大量的存储空间，一个普通物体的八叉树就可能多达数千个节点，而每个节点又需要8个指针域，开销相当昂贵。因此，可以使用一种经过编码优化的一维数组来存储八叉树，这里不多赘述。

IV. 多项式和开花【黑书19】

多项式是参数化构型的基础，只有对多项式的性质进行深刻的讨论，才有可能建立起来一个缤纷多彩的世界。

一个多项式一般可以写成Σat^n的形式。对于线性变换f(t)来说，由于f是一个仿射变换，所以可以保持在线性映射中的不变性。考虑到t=(t2-t)/(t2-t1)\*t1+(t-t1)/(t2-t1)\*t2，进而得到f(t)=(t2-t)/(t2-t1)\*f(t1)+(t-t1)/(t2-t1)\*f(t2)。

接下来，试着把这种仿射变换的性质推广到多个参数上。定义一个多仿射变换是一个关于t1,t2,...,tn的多元函数，并且这个函数只含有常数项、一元一次项、二元二次项、……等项的线性组合。比如，三变量的仿射映射就是f(t1,t2,t3)=c0+c1t1+c2t2+c3t3+c4t1t2+c5t1t3 +c6t2t3+c7t1t2t3。

从此之上，可以引入开花的概念。任意一个多仿射映射一定有且仅有一个多项式与之对应，这就是开花定理。实际上，对于k元的仿射变换，假设t^q项的系数是c(q)，那么对应到开花的多重映射中，每一项的系数都是c(q)/C(k,q)。

V. Bezier Curve【黑书19、白书9】

先从de Casteljau算法的角度给出对Bezier Curve的定义。假设有四个控制点分别是f(r,r,r)、f(r,r,s)、f(r,s,s)、f(s,s,s)，那么按照层级进行插值运算求解f(t,t,t)的过程实际上就是构造Bezier Curve的一个点的过程。具体来说，对于四个点P1, P2, P3, P4，首先在P1P2, P2P3, P3P4上找到t所对应的点P1’、P2’、P3’，再继续从P1’P2’、P2’P3’上找由t线性组合对应的点P1’’、P2’’，最后从P1’’P2’’线性组合出P1’’’，就是目标点。

容易证明，Bezier Curve实际上具有一定的数学形式：F(t)=ΣB(n,i)(t)p，其中B(n,i)是BernStein基函数，B(n,i)(t)=C(n,i)t^i(1-t)^(n-i)。这种方式构造出来的实际上是对所有控制点的一个线性组合，但具有非常良好的性质。

这条曲线的端点是左右两个控制点，端点处切线正好和P1P2、Pn-1Pn平行。曲线具有凸包性，也就是曲线位于控制顶点构成的凸包之内。如果想要改变曲线形状，只需要修改控制点，所以这种曲线很适合用来交互。控制多边形的凸性可以影响曲线，也就是曲线的凸性和多边形保持一致，叫做保凸性。除此之外，曲线和控制点连线的交点个数不多于直线和控制点多边形的最多个数，也就是变差缩减性。

想要生成一条Bezier Curve，使用de Casteljau方法或是公式展开都比较复杂，常常使用一种离散生成的方法。这种方法是基于一个递归性质：对0<=t<=1/2和1/2<t<=1，P(t)=Σ P(i)J(i,n)(t)，在0<=t<=1/2的时候P(t)=ΣPi[i]J(i,n)(2t)，1/2<=t<=1的时候P(t)=ΣPn[n-i]J(i,n)(2t-1)。其中Pi[r]在r=0时为Pi，否则为(Pi[r-1]+Pi-1[r-1])/2。从几何意义上看，就是不断对多边形进行裁切，最终得到一个对贝塞尔曲线的逼近。

在实际开发中，一般使用的都是三阶的Bezier Curve。这就涉及到了连续性的问题，如何把不同的Curve拼接到一起的时候，还能维持连续性呢？在微分几何中，一般有几种常见的连续性定义。在数值上连续叫做G0连续，如果切向相同叫做G1连续，如果切向相同且大小相等叫做C1连续，类似的可以定义二阶导数的几何连续G2连续等。对于Bezier Curve，只要控制点首尾相接就有G0连续，如果控制点的首尾向量平行就有G1连续。

VI. B-spline【黑书19、白书9】

B-spline就没有优雅的几何形式了。和Bezier Curve一样，它也是对一系列函数的线性组合，所以先给出其基函数。B-spline的基函数这样定义：B(i,k)(t)=(t-t(i))/(t(i+k-1)-t(i))B(i,k-1)(t)+(t(i+k)-t)/(t(i+k)-t(i+1))B(i+1,k-1)(t).它的基函数是一个分段函数，只在[t(t),t(i+k)]的范围内取正数，其他地方都是0，并且对同阶的基函数，其权之和为1.

那么，B-spline就是ΣPiB(i,k)(t).同样的，B-spline也是从一系列控制点出发的样条曲线，这条折线构成控制多边形。同样的，它也具有保凸性、变差缩减性、连续性，但是并不一定过两个控制多边形的端点。

渲染一条B-spline一般使用de Boor算法，也就是建立一个类似的三角形关系进行递推。B-spline可以使用Oslo算法进行插入操作，具有良好的可操纵性。另一方面，l重节点的连续阶不低于k-1-l，这使得曲线有非常好的连续性。

VII. 参数化曲面【黑书19、白书9】

也可以用Bezier Surface或B-spline surfaces完成曲面的参数化。

对于Bernstein基函数表示的曲面，一般有形式F(t,u)=ΣΣB(t)B(u)p。如果交换球和指标，实际上可以看成对一系列Bezier Curve进行的求和，并且分配的权重是基于Bernstein基函数。这也给了另一种看待Bezier Surface的方式。

这种参数化曲面也有很好的性质。它的四个端点正好是控制平面的四个端点，边界线正好是四个边界控制点构成的Bezier Curve。它端点的切平面恰好是位于角落的四个三角形所在平面，并且有凸包性，即曲面落在构成的控制平面内。用切线上的连续性也很容易实现曲面拼接。

使用B-spline的基函数，也可以定义B-spline surfaces。它也构成对控制曲面的一个逼近，并且有很好的连续性。

如果给定一个曲面，想要做一个端点、端点导矢、端点扭矢都相同的多项式曲面，可以使用Hermite基函数作双三次多项式插值，得到的就叫做双三次孔斯曲面。孔斯曲面可以和Bezier Curve进行互化，有许多有趣的性质，这里不过多展开。

五、动画

动画，或者说模拟，一直以来是研究的一个热点。这个领域是图形学最广为人知的地方之一：从受欢迎的迪士尼卡通角色，到电影中的各种特效，这些动画领域都在不同行业得到了自己的应用。有些动画则致力于研究如何模拟出最真实的物理效果，比如布料和流体。

基于物理的动画是动画研究的核心领域，其中最简单的系统是质点弹簧系统。在模拟一个有骨骼和关节的系统的时候，低层运动控制实现容易，但是比较复杂；反向运动学则大大简化了这一过程。粒子系统可以用来模拟大量粒子的群体性行为。在模拟形变的时候，常常使用Morphing技术。

I. 动画的发展【白书12】

计算机图形学和艺术的联系总是非常紧密。动画是一个和艺术关联性非常大的领域，它的目标就是创造出一个天马行空的动画世界。1993年，《侏罗纪公园》上映，其中利用了许多计算机特效，让人眼前一亮。1996年，《玩具总动员》上映，这是第一步完全由计算机动画制作的电影，带来了空前的票房。

不论是传统卡通动画还是现在的动画，其原理都是人类的视觉暂留效果。起初，由于在关键帧之间，需要大量绘制中间帧，人力成本很大，所以计算机用来做插值来完成补间操作。这项技术在1964年的贝尔实验室被首先尝试，也标志着新时代的萌芽。

70年代，随着图形学的基础理论得到了飞速发展，光照模型、纹理映射、三维造型的技术都被发明出来，80年代有大量商业动画软件诞生。从这个时代开始，动画大放异彩，得到了广泛应用。Flash技术让动画走进了千家万户，变成了计算机发展史中最富有动感的元素之一。

如今，各个领域都无法摆脱计算机动画。在广告业界，动画可以做出非常夸张的表演，吸引人的注意；在影视业界，动画不仅被广泛应用在特效设计中，同时自己也成为艺术题材的一种；在工业界，传统的几何造型是静态的，而动画则让装配成为可能。

II. 低层运动控制方法【白书12】

所谓的低层运动，主要是对底层参数、而不是行为本身进行控制。不论是低层控制还是高层控制，都最终作用于参数上，但低层控制需要人为操控复杂的数据。

关键帧最早用于运动控制。这个概念从早期动画中就得到了普及，最早用在帧和帧的插值中。具体来说，给出一系列关键的参数，然后在时间轴上对这些参数进行插值，最糟得到了一系列的离散参数值，就可以实现运动控制。

如果先设置好运动轨迹，就可以按照这个轨迹进行动画。一般在这个过程中以弧长作为参数，按照固定弧长来进行迭代积分。很多时候用一条曲线来表示运动状态，也就是弧长对时间的导数变化。

在旋转问题中，常常使用四元数来完成插值操作。具体来说，就是借助Slerp(q1,q2,u)=sin(1-u)θ/sinθq1 + sin uθ/sinθ q2，来当u∈[0,1]，得到一系列的值，作为插值结果。如果在多个关键帧使用插值的时候需要保持比较高的连续性，可以将一般样条曲线推广到四元数空间中。

III. Morphing和空间变形技术【白书12】

在动画中，常常看到一个物体逐渐变成另一种，这就是Morphing技术的来源。1982年，Brigham制作过一个短片，从一个女人变成了一个山猫，这是最早的一系列工作之一。Morphing是指从一个状态逐渐过渡到另一个状态，要求中间帧保持关键帧的特性，需要指定源对象和新对象的对应关系。空间变形则是在保持拓扑结构不变的情况下发生某些扭曲，产生新的物体。

对于二维多边形，其变换涉及到两个问题：顶点的对应关系，顶点的插值。假如f(t)表示从A到B的变换，且f(0)=A, f(t0)=B，那么最简单的方法就是在两个数集上构造线性插值，但是容易出现收缩和扭结现象。可以对夹角和边长进行插值，达到到比较好的效果。

二维图像是一个比较复杂的技术，但是十分有用。这个过程需要首先计算出一些几何元，比如网格节点、线段，然后构造两个满射，分别把第一个图映射到第二个图和第二个图映射到第一个图。然后把两个漫射做线性插值之后再进行一次颜色插值，就得到了比较逼真的效果。网格形变是最早应用的形变技术，但是比较复杂，1992年引入了基于线对的特征指定方法，通过一定变换让直线发生变换。

三维物体有了更多的自由度，能得到非常逼真的效果。如果两个多边形拓扑结构相同，可以很容易进行插值，但是如何找到这种同构性？Kent等人提出了合并拓扑结构，先将物体投影到单位球上，然后再将拓扑结构合并，最后映射回来。Lerios等人则提出了用体表示实现的三维morphing。

如果将物体嵌入到一个空间中，使用参数曲面对空间变形，就可以让物体也随之发生形变。对于一个三维物体，可以使用Bezier超曲面，构造Bezier体，形成FFD块，实现自由变形。这种方法就是FFD方法。

元球是一种特殊的隐式曲面，很适合用来描绘光滑物体。多个元球之间相互融合的过程，可以刻画出比较复杂的动画，具有光滑性的同时也节省了大量空间。

IV. 过程动画【白书12】

过程动画是用一个过程来描述物体的运动和形变。这种过程常常基于数学模型或物理规律，体现出复杂性。

粒子系统是把许多微小粒子作为基本元素，构建出一个复杂的系统，在这个系统中对大量粒子的统计规律进行模拟，实现许多复杂的效果。粒子有生长和死亡过程，在生命周期中不断发生形变。Reeves专门对粒子系统给出了一个绘制算法，假设不和几何模型相交，然后对覆盖像素的光亮度贡献是点光源，达到了比较高的效率。

群体动画是对动物群体的运动进行描述的方法，即相互靠近同时避免碰撞。Reynolds给出了三条定律来匹配群体运动。

布料渲染一直是一个热点话题。最早的布料是通过悬挂在一些约束点上利用悬链线来计算，但这种方法不能模拟褶皱。现在一般使用物理方法进行模拟，考虑物理材料的各种参数。

水体渲染最简单的方法是使用一个正弦波来进行参数设置，也可以使用一些波动理论来模拟传播过程。

V. 骨骼动画【白书12】

如果把一个人的身体抽象，会发现控制运动的实际上就是许许多多的关节。同样，如果对人体模型绑上骨骼，那么对这些骨骼的控制实际上就转换成了对关节的控制。在这个过程中，只需要关注它的运动学模型和动力学模型。

在一个关节链结构中，刚体的连接点是关节，连接关节的刚体是链杆，起点是根节点，末端是末端影响器。关节链结构的可能形态向量空间是结构的状态空间，由系统自由度决定。将许多关节链结构有约束连接，就构成了复杂模型。

可以使用DH表示法来描述一个关节链结构，它由链杆长度、相邻链杆距离、扭角、夹角四个参数构成。对关节链的控制可以使用正向控制，也可以使用反向控制。对于一个骨架系统，常常构造一颗骨架树来描述关节链之间的关系。在骨架树中，如果要进行运动控制，可以使用一对对节点来构造单链，在树上更新。这样逆向运动学就能够推广了。

如果在骨架上建立肌肉和皮肤，就可以实现更丰富的效果。比如脸部的表情动画，就可以借助肌肉的变化来营造出各种表情。

VI. 物理动画【白书12】

传统的动画想要模拟一个真实的物理系统，需要动画师对生活的准确把握。而如果按照物理规则进行推演，就可以实现比较逼真的效果。

刚体运动是最常见的场景，在运动过程中通过碰撞检测和响应就可以实现刚体系统。比如，用动力学方程模拟关节链运动，可以让动画和场景发生交互。其中最核心的问题就是碰撞检测的问题，包围盒、八叉树等场景构造方法都可以加速检测过程。

质点-弹簧系统常常被用来描述塑性物体的形变。Weil将其用在布料在钉子上悬挂的过程，然后被推广到各个领域中。也有使用连续弹性系统来实现物理模拟，或引入振动系统。

流体力学作为力学的重要分支，在图形学中一般使用恰当的流体运动偏微分方程，求出数值解，就可以得到流体的形状和位置变化。

六、结语

图形学的世界丰富多彩。将一些看上去仅仅是数据的东西呈现在屏幕上，这本身就是一种创举，更是一个新的世界。在这个世界中，无数的人通过全新的方式表达自己，也带来了如今丰富多彩的媒体世界。

无论是渲染、几何还是动画，都构成了这个五彩斑斓世界的一部分。有的人为了在虚拟的世界中营造真实感而拼尽全力，有的人为了构造出一张完美的曲线而不断努力，我想这都是图形学的魅力所在。图形学是一个性价比很低的学科，因为投入很多，产出却往往不太匹配。但只要有人真正爱着图形学，这个世界就会更加的五彩斑斓。