

可视计算与交互概论课堂笔记

信息科学技术学院 Seto1

2025 秋

前言

Seto1

2025 年 11 月 23 日

目录

第一章 基础知识	1
1.1 颜色	1
1.2 显示	2
1.3 画图	2
1.4 曲线	4
1.5 反走样	6
第二章 图像处理基础	8
2.1 图像抖动与量化	9
2.2 图像滤波	10
2.3 图像修复/融合	12
2.4 图像拼接	13
2.5 图像压缩（图像频域变换）	14
第三章 不那么基础的图像处理	16
3.1 图像插值与超分辨率重建	16
3.2 图像去噪	18
3.3 图像直方图与低光照增强	22
第四章 三维重建	26
4.1 极线几何	27
4.2 双目立体视觉	28
4.3 运动恢复结构 (SfM)	30
4.4 MVS	31
第五章 几何	32
5.1 几何表示	32
5.2 几何处理	33
5.3 几何重建	35

5.4 几何变换	39
第六章 图像生成模型	41
6.1 变分自编码器	41
6.2 生成对抗网络 (GAN)	43
6.3 Transformer	45
6.4 扩散模型 (Diffusion)	46
第七章 渲染	49
7.1 光照与着色	49
7.2 渲染管线	51
7.3 纹理映射	52
7.4 光线追踪	53
7.5 路径追踪	55
第八章 仿真	57
8.1 物理模拟	57

第一章 基础知识

1.1 颜色

颜色是人类观察世界的基础。显色原理主要分为增色（光源显色，即 **RGB**）和减色（反射吸收光显色，即 **CMY (K)**，青-品红-黄-（黑），在打印中应用较广）。

人类感色主要依赖视网膜中央凹的**视锥细胞**，其分为 S/M/L 三种，分别对应蓝、绿、红波段的敏感性，缺失其中的若干种会导致色盲，而 S-视锥细胞较少也导致人类对蓝光普遍不敏感。但人类对物体形状的感知并不主要依赖感色，而是亮度通道（明暗差异），于是我们可以通过等亮度调节减弱边缘层次感，营造漂浮感。

人类感色还有一些特性：由于不同波长的光在视网膜成像时焦点位置不同，导致形成**色彩立体视觉**，即红色在视觉上更靠前，蓝色更靠后；大脑会进行颜色的**恒常性修正**，使物体在不同光源下颜色保持相对稳定（如白纸始终被感知为白色）；同时，周围颜色会对感色产生影响，例如**同时对比现象**（灰色在黑色背景下显得比白色背景下更亮）；**色诱导**（颜色会被周围颜色推向对比色）；**边界效应**（边框的颜色会影响对内部颜色的感知）。

计算机中表示颜色的空间主要有 **RGB 空间**（朴素的 $1*1*1$ 立方体）和 **HSV 空间**（色相-饱和度-明度，可以更方便地根据人类需求调节，其通过**柱坐标形式**呈现，可与 RGB 空间一一对应）。有时还会运用 **Lab 空间**（亮度-红至绿-蓝至黄）、**YCbCr 空间**（亮度-蓝色分量-红色分量）等，可以看出，优化后的色彩空间都在强调人眼敏感的亮度调整。与 HSV 空间类似的还有 **HSL 空间**（L-亮度，不同于明度 $V = \max(R, G, B)$ ，亮度 $L = \frac{R+G+B}{3}$ 时必定为白色），二者在图像处理中均有应用，其与 RGB 的转换公式如下：

■ RGB \leftrightarrow HSL

$$H = \begin{cases} \theta & B \leq G \\ 2\pi - \theta & B > G \end{cases}$$
$$\theta = \arg \cos \left(\frac{2R - G - B}{2\sqrt{R^2 + G^2 + B^2 - RG - RB - BG}} \right)$$

$$S = 1 - \frac{3 \min(R, G, B)}{R + G + B} \quad L = \frac{R + G + B}{3}$$

■ RGB \leftrightarrow HSL

	$0 \leq H < 2\pi/3$	$2\pi/3 \leq H < 4\pi/3$	$4\pi/3 \leq H < 2\pi$
	$H' = H - 2\pi/3$	$H' = H - 4\pi/3$	
R	$L \left[1 + \frac{S \cos H}{\cos(\pi/3 - H)} \right]$	$L(1 - S)$	$3L - (G + B)$
G	$3L - (R + B)$	$L \left[1 + \frac{S \cos H'}{\cos(\pi/3 - H')} \right]$	$L(1 - S)$
B	$L(1 - S)$	$3L - (R + G)$	$L \left[1 + \frac{S \cos H'}{\cos(\pi/3 - H')} \right]$

为了更好地与人类视觉对齐，也开发了一些相关技术：**HDR**（高动态范围）技术通过多次曝光，加大图像中最亮和最暗区域的比值，使高光和阴影细节都能保留，更接近人眼所见（如拍摄落日）；而**Gamma 校正**基于人类对暗处细节更敏感的特性，利用类似 $L = V^\gamma$ 的公式刻画输入信号与显示亮度的非线性关系，使其更符合人眼感知。

1.2 显示

简单的二维显示技术可以直接使用**数码管显示器**，即户外大屏使用的集成大量 LED 点阵的显示器，但这难以显示复杂图像。20 世纪电视显示依赖的主要是**阴极射线显示器 (CRT)**，其可通过发射电子束在荧光屏上进行**扫描**，进而成像。扫描主要分为**光栅扫描**（从左上到右下按顺序扫描）和**随机扫描**（在绘制简单矢量图时，按照命令绘制图像）。其中，光栅扫描又可分为逐行扫描 (P) 和隔行扫描 (I，利用视觉暂留，奇数行与偶数行交替显示，解决带宽不足问题），例如常见的 480p60 就是指的**总共 480 行，逐行扫描，帧率为 60 次/秒**。这类技术在 21 世纪已逐渐被液晶显示屏 (LCD) 和有机发光半导体显示器 (OLED) 取代，目前的二维显示已经发展至 8K 超高清 (7680*4320)，色域不断扩张，并利用 HDR 技术扩展亮度范围，利用**超分辨率显示技术**突破单屏分辨率制造成本限制等。

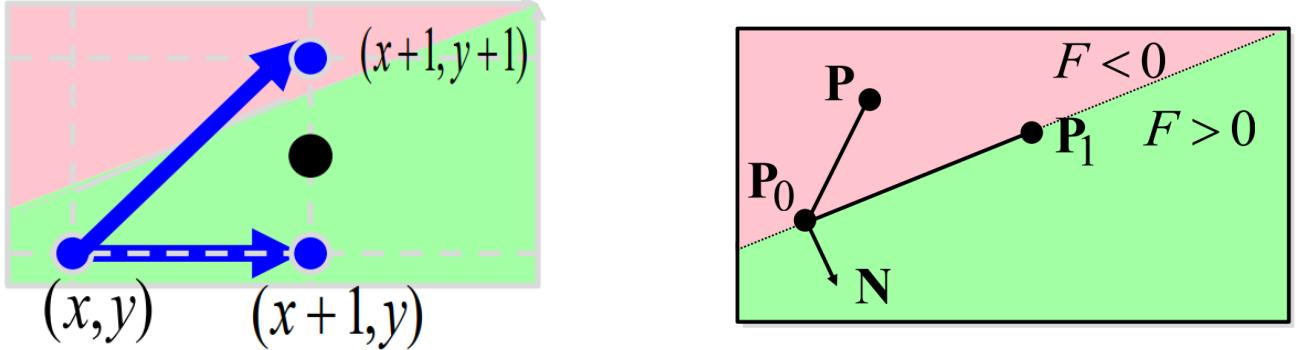
三维视觉主要是利用**人为增强人类双目的视差**，制造立体感。常用的 3D 眼睛是通过左右镜片偏振方向不同创造的立体视觉，而裸眼 3D 则是直接用屏幕结构（如微柱透镜、光屏障等）或动态检测人眼位置调整屏幕（动态裸眼），让左右眼接收到不同图像。**全息显示**则试图在亮度外额外纳入**光的相位**的考量，力求直接还原自然光波。

1.3 画图

画图这一话题从画直线开始，我们需要设计一个既准确又高效的算法以绘制直线。朴素的想法是：不妨设直线斜率 $k \in [0, 1]$ ，则只需扫描 $[x_0, x_1]$ ，计算出每个 x 对应的 $y = \text{Round}(kx + b)$ 即

可，进一步可以通过 $y_{x+1} = y_x + k$ 优化掉繁复的浮点数乘法运算，获得朴素的 DDA 算法。

但这样的效率仍然太低，注意到（在 $k \in [0, 1]$ 时）每次的下一个点只能是 $(x + 1, y)$ 或 $(x + 1, y + 1)$ ，具体选取哪种取决于 $(x + 1, y + \frac{1}{2})$ 与直线的位置关系：



而该问题可以进一步转化为上右图中的 $\overrightarrow{PP_0}$ 与法向 $\vec{N} = (dy, -dx)$ 的夹角是否大于 90° ，等价地，只需考察点积

$$\vec{N} \cdot \overrightarrow{PP_0} = (x - x_0)dy - (y - y_0)dx = F(P)$$

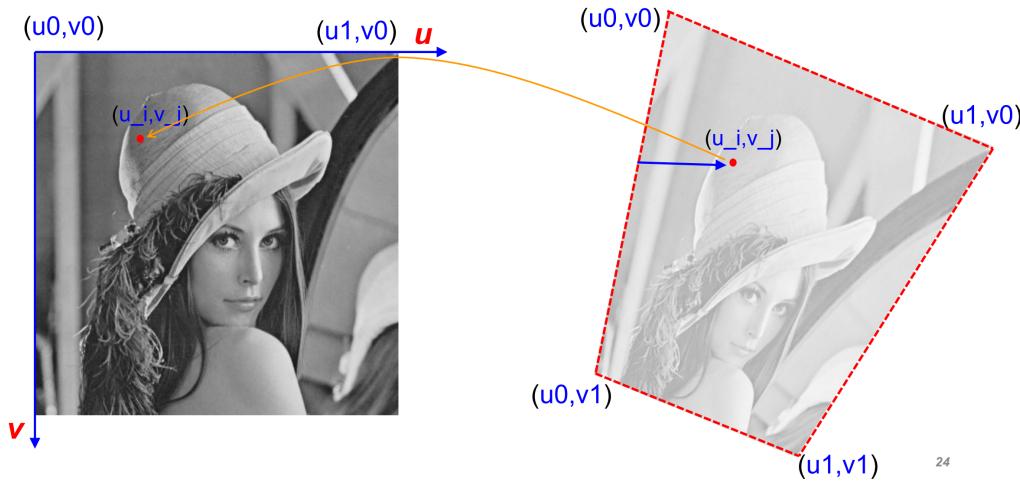
的符号，我们可以记录每个 x 对应中点处计算得到的 $F(P)$ ，并通过如下递推关系更新结果：

$$F(P_{x+1}) = \begin{cases} F(P_x) + dy & F(P_x) < 0 \quad (\text{即中点在上方, 水平移动, 同时 } x \rightarrow x + 1) \\ F(P_x) + dy - dx & F(P_x) \geq 0 \quad (\text{即中点在下方, 斜向移动, 同时 } x \rightarrow x + 1, y \rightarrow y + 1) \end{cases}$$

其中 F 初值置为 $F(P_0) = dy - \frac{1}{2}dx$ ，为了方便整数计算，过程中所有值乘 2 处理即可。需要注意，开始时我们作了 $k \in [0, 1]$ 的假设，在实际计算中需要综合考虑各个斜率的情况。这种绘制直线的方法便称为 **Bresenham 直线算法**，将其推广至圆的隐式方程 $x^2 + y^2 = r^2$ ，也可同样地绘制出圆。

进一步地，可以得到绘制多边形的**扫描线算法**，其核心是找到每条水平扫描线对应的左右边界（这个过程可以通过 Bresenham 直线算法实现），并在**统一的边界取整规则**的前提下绘制扫描线（否则会出现裂缝或凸起），对于凹多边形也可以用相同的操作，在凹点附近可以用通过变量记录内部-外部的翻转实现。在实际实现中，也可以使用**三角化方法**（这是因为 GPU 内部有专门资源对三角形进行渲染），通用的三角化算法是 **Ear Clipping**，也即重复【每次找到一个凸点，将其对应的两条边剪掉】直到剩下一个三角形。

推广到彩色图像中，我们通常需要通过图形顶点的颜色绘制图形内部，此时在扫描线的基础上需要进行**颜色插值**，具体操作为先在左右边界上插值，再在扫描线上插值得出具体点的颜色。插值技术不仅能应用于颜色绘制中，也可以应用于**图像变形**中，如下图（同样运用了扫描线）：



最后，当图像中存在透视关系（即顶点间存在深度差异），并且对图像进行了非线性变换（不仅包括平移、旋转、缩放等线性操作，而是涉及透视投影或除以深度 z 的映射）时，就需要在插值阶段额外进行**透视校正**，以保证纹理或颜色的空间一致性，这会在后续的纹理映射一节中涉及。

1.4 曲线

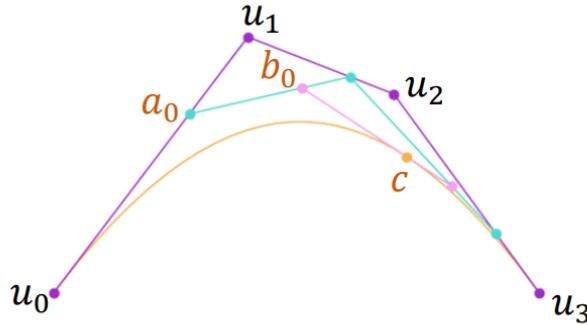
对于曲线的表示大致可分为三种：**显式表示** ($y = f(x)$)，**隐式表示** (如 $x^2 + y^2 = r^2$)，以及**参数式表示** ($(x, y) = (f(t), g(t))$)。其中，参数式表示因其易简单推广至三维、曲线可用向量表示、容易计算导数（即切向向量）等优点，被广泛研究。在下面的内容中，就将聚焦用参数（尤其是多项式）表示简单的二维曲线。

理论上来说，可以通过 $n - 1$ 次多项式拟合 n 个点确定的曲线，但同样显然，这样的效果十分不好，因此我们通常将曲线进行**分段**，在小段（即点数较少的情况下）进行处理。

从最简单的情况看起。若只有两个点，显然其间的“曲线”就是连成的直线，其方程可以用下面的**线性插值**表示：

$$\text{lerp}(t) = (1 - t)u_0 + tu_1$$

推广到多个点的情况，实际上我们也只需要**反复进行线性插值操作即可**。具体而言，按顺序在第 i 和第 $i + 1$ 个点之间插值出一个新的点，总计得到 $n - 1$ 个点，重复上述操作直到只剩一个点，该点的轨迹就是这 n 个点确定的平滑曲线，也叫**Bezier 曲线**（它是 SVG 矢量图绘制图形的重要数学基础之一），这种求 Bezier 曲线的方法被称为**De Casteljau 算法**。下图即展示了四个点的情况：

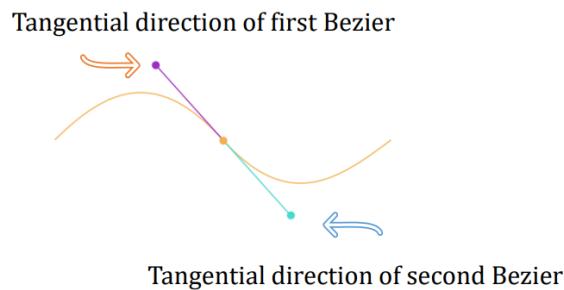


还可以从另一个视角观察插值的过程：与线性插值 ($u = (1 - t)u_0 + tu_1$) 过程类似，最终的点可以看作控制点的一个线性组合，观察 D-C 算法，线性系数（也即每一个控制点的权重）是显然的：

$$b(t) = \sum_{k=0}^n \binom{n}{k} t^k (1-t)^{n-k} u_k$$

或者写成矩阵形式： $b(t) = B(t)^T u$ ，这其中的向量 $B(t)$ 便被称为**基函数**。

接下来我们需要将 Bezier 曲线拼接成完整的曲线，这就对连接处的光滑性提出了要求。为了保证这一点，我们需要调整控制点，使两端曲线在端点处相邻的“控制向量”等大反向（这是因为经过计算，曲线首尾的切线方向恰是其对应“控制向量”方向，具体如下图）：



但这依赖于人工控制且调整一个控制点可能导致多条曲线的改变，而 **Spline (样条曲线)** 可以通过在数学上引入连续性约束方程，使曲线在各段拼接处自动保持平滑，从而解决这一问题。根据对连续性要求的不同，样条曲线可以进一步分为多种类型：若只要求 C^1 连续，可以使用 **Hermite 样条**（通过端点及其导数调整参数，从而保证光滑）或是 **Catmull-Rom 样条**（将 Hermite 样条中的导数用控制点间割线斜率拟合，避免手动设置导数）；若要求 C^2 连续，可以使用**自然三阶样条**，通过每两点之间拟合一个三次多项式 $S_i(x) = a_i + b_i x + c_i x^2 + d_i x^3$ ，并通过端点处 C^0, C^1, C^2 连续进行约束，如此得到的曲线光滑性得以强制保证，但全局耦合的问题仍旧存在。为完美解决如上问题，最终提出 **B-Spline**，用此前提到的基函数思想解决这一问题。

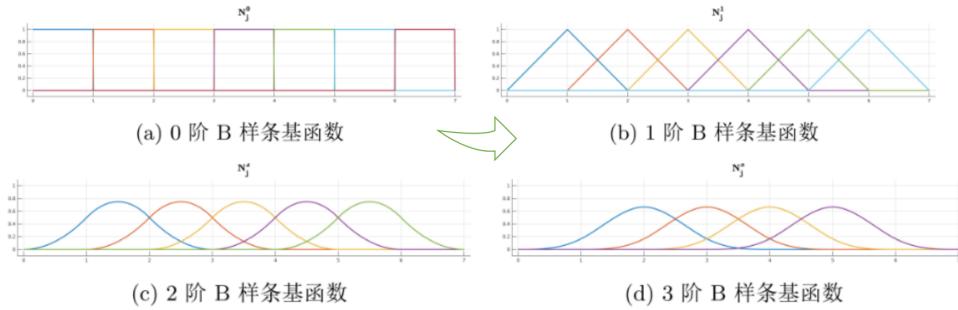
B-Spline 的方程形式与 Bezier 曲线的颇为相似： $b(t) = \sum_{i=0}^m N_{i,n}(t) P_i$ ，其中 $m+1$ 表示点数，

n 表示阶数，而 $N_{i,n}$ 表示基函数，其由如下的递推公式给出：

$$N_{i,0}(t) = \begin{cases} 1, & t_i \leq t < t_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

$$N_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} N_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} N_{i+1,p-1}(t)$$

具体形状如下图所示。每升一阶，相当于增加一个控制点（影响范围 +1）；而递推公式代表了两个较小范围基函数的线性组合，通过设计系数实现了控制点系数的归一化。可以发现， $N_{i,p}(t)$ 的导数是 $N_{i,p-1}(t)$ 和 $N_{i+1,p-1}(t)$ 的线性组合，于是每升一阶，相当于增加一重光滑性。在行列两个方向分别做控制点加权，也可以实现二维的 B-Spline。

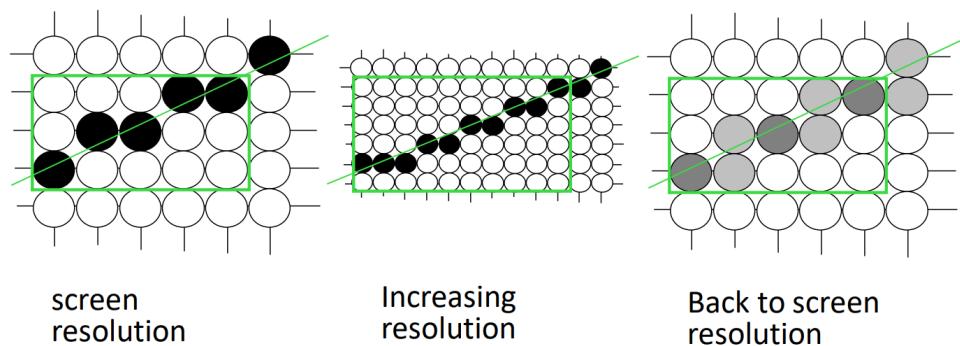


1.5 反走样

离散的采样会带来锯齿状的“走样”，无法表示连续的几何变化，于是需要开发反走样算法解决这一问题。

首先尝试转换视角，从频域角度出发理解一下走样（对于频域的介绍见下图像压缩一节）。根据傅里叶变换的性质，在时空域中采样（即对于原函数乘一个冲激列）等价于函数与冲激列作卷积，也即讲原波形复制为无穷个副本，每隔 f_s 复制一次。于是当采样频率过低时，复制的波形之间会产生重叠，从而形成走样。这同时也给出了解决方案：去除高频分量，即进行低通滤波，使图像变平滑。

具体而言，先增大分辨率，在高分辨率下作低通滤波，再将高分辨率结果平均（整合）为目标像素，从而实现反走样，这就是超采样反走样（SSAA）算法。效果如下图：



事实上，SSAA 也有一种简单的替代方法：若我们计算各像素点到直线的距离，根据距离富裕不同的灰度，也可以实现类似 SSAA 的效果。

走样现象不仅出现在画线中，也出现在图像变形的过程中。经过透视变换后，在远处区域，一个像素可能包含很多原图信息，由于**欠采样**，就会出现锯齿等走样现象。由此我们需要推出针对此的反走样算法——**MIP mapping**。

其核心思想是提前预算算不同分辨率的纹理版本（一系列“模糊版”纹理），并在渲染时根据需要选择最合适的一层。该预处理通常可由高斯金字塔的下采样过程生成。接下来要解决的问题是如何选取合适的层级。理想情况下，应当让**金字塔的分辨率与屏幕分辨率尽可能匹配**：

$$L = \max \left(\sqrt{\left(\frac{du}{dx} \right)^2 + \left(\frac{dv}{dx} \right)^2}, \sqrt{\left(\frac{du}{dy} \right)^2 + \left(\frac{dv}{dy} \right)^2} \right)$$

随后即可通过 $D = \log_2 L$ 得到对应层级，并在 $\lfloor D \rfloor$ 与 $\lceil D \rceil$ 两层之间进行线性插值。

第二章 图像处理基础

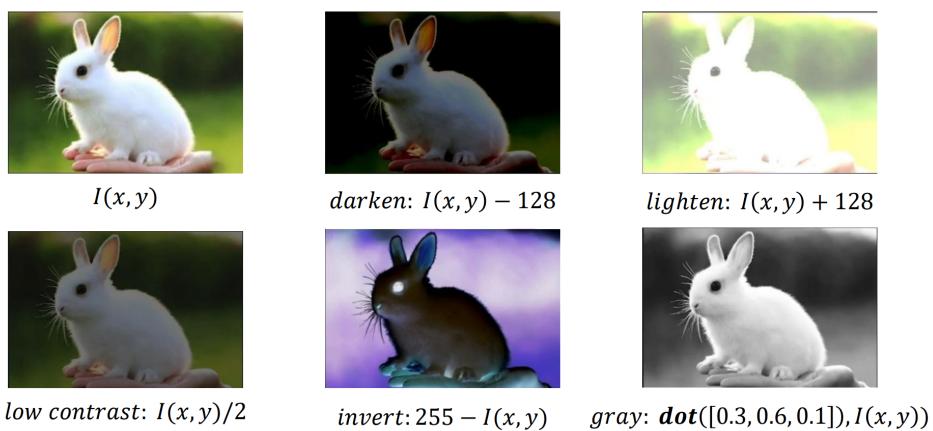
先补充一些图像显示的基础知识。

图像的显示。位深度（8位 =255， 灰度级：表示像素明暗程度的整数量）常见的图像格式主要包括：**BMP**（Windows 位图，无损，无编码压缩，可直接读取像素值）/**PNG**（无损压缩位图格式，可表示透明图像），**JPEG**（有损压缩，图片质量可根据压缩的设置而有所不同）

如果需要显示多个图层，则需要在 RGB 的基础上额外考虑透明度 α ，例如两张图像合成就可以利用 $b = \alpha a_1 + (1 - \alpha)a_2$ ，同时为了表示远近遮挡关系，还需要存储 **Z-buffer**（深度缓存），使得系统自动保证只显示最靠前的物体。同时，为了支持动画无闪烁，还使用了**双缓存**，分为前台（显示中）与后台（正在绘制），绘制完一帧后，交换两者。总计需要 $【8*3\text{ (RGB)} + 8\text{ (透明度)} + 16\text{ (深度)}】 * 2 = 96$ 比特/帧。

接下来正式进入图像处理部分，最主要的两种操作是点处理和**邻域滤波**（filtering）（即输出像素取决于该像素及其周围邻域的像素值）。点处理就是指直接对像素值 $I(x, y)$ 进行逐像素操作，部分经典操作如下图所示：

Pixel (Point) Processing



其中最经典的就是模式为 $g(x) = af(x) + b$ 的变换，其中 a 代表调节对比度， b 代表调节亮度。同时，**幂律变换** $y = cx^\gamma$ 会在 $\gamma > 1$ 时使图像整体变暗，且增强亮部对比度，而在 $0 < \gamma < 1$ 时取得相反效果。

图像之间也可以进行简单的算术运算，例如图像相加可用于加噪，相减可用于比较，相乘可用于蒙版，相除可用于消除光影。

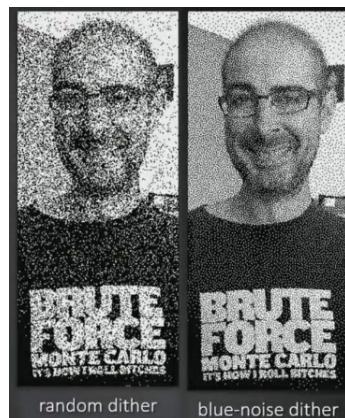
除了像素级操作外，还包括几何变换、压缩编码、语义分析等更高层任务。以下重点介绍几种图像处理任务：

2.1 图像抖动与量化

图像的量化（Quantization）是将连续的像素值离散化为有限个等级，从而减小文件大小的过程，最朴素的量化（以灰度图为例）是把灰度值简单地按 0.5 为阈值分为黑白二值，但这样的图像显然并不自然（下右图）：

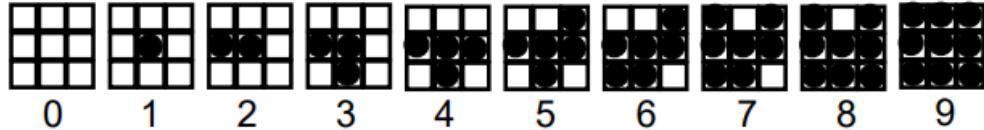


于是考虑在量化前向像素值中加入随机扰动（噪声），使误差不再**系统性集中**，而是被“打散”成随机噪声，最终使得人眼看起来更平滑（人眼对高频信号不敏感）。这就是**图像的抖动（Dithering）**。常见的噪声包括**白噪声**（完全随机，呈现雪花状）和**蓝噪声**（保留随机性，但特意抑制低频成分、强化高频分量）。显然蓝噪声更适合完成图像抖动任务，其实现相对复杂，包括后面会提到的**频域滤波法**以及 **Poisson Disk Sampling**（保证在每个点半径为 r 圆盘中没有其他点的前提下实现最大填充）。两种噪声的效果图如下：

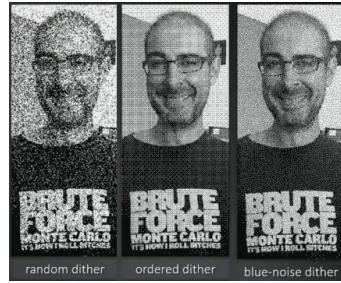


另一种抖动的方式是**空间抖动**，即放大图片，用多个黑白点的排列方式来“模拟”中间亮度，从而实现用空间换分辨率。要实现上面的效果，我们需要定义一套**点阵模式**，以说明不同亮度下哪些像素是亮、哪些是暗，我们可以用一个 3×3 的**抖动矩阵**表示这一过程，例如 $M = \begin{pmatrix} 6 & 8 & 4 \\ 1 & 0 & 3 \\ 5 & 2 & 7 \end{pmatrix}$ ，

输入亮度会依次激活不同数量的点，如下图：



最终的效果图如下（下图中）：



最后一种方法采用了完全不同的思路：它不是将误差打散，而是通过**扩散**意图某种程度上“消除误差”。其实现过程是：按顺序扫描像素，在简单二值化分类的基础上计算误差 $\text{error} = I[x, y] - O[x, y]$ ，并将误差按照一定比例**扩散**到后续像素点上（在 Floyd-Steinberg 算法中，比例被经验地设定为【右: $\frac{7}{16}$; 左下: $\frac{3}{16}$; 下: $\frac{5}{16}$; 右下: $\frac{1}{16}$ 】）。

以上各种算法可以自然迁移到 RGB 三通道上。

2.2 图像滤波

图像本身可以看作一种波(信号)，对其进行某种过滤操作就被称为**图像滤波**(image filtering)。在数学上，图像滤波通常通过**卷积**来实现。对于一维信号 $a[x]$ ，其滤波结果可以写作

$$b[x] = \sum_{t=-\infty}^{+\infty} a[t] h[x-t],$$

其中 $a[x]$ 为输入信号， $h[x]$ 为滤波器（卷积核）， $b[x]$ 为输出信号。在二维情况下（图像），卷积扩展为双重求和形式：

$$b[x, y] = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} a[u, v] h[x-u, y-v].$$

在实际应用中，卷积核是有限大小的，也即：

$$b[x, y] = \sum_{u=-p}^p \sum_{v=-q}^q a[u, v] h[x-u, y-v]$$

其中 p, q 一般取为奇数以保证对称性。需要澄清的是，严格意义上来说，图像滤波应当做的是卷积运算（其特点是滤波器需要进行翻转）而非不需要翻转的**相关性运算**，但是实际上在计算机实现中，滤波器大部分是对称的，因此直接使用相关性运算（correlation）即可。

* 相关性运算应用广泛，经典一例便是模板匹配，即在图像中识别出特定的（待补）

同时，还需要注意**边界效应**（滤波器超出图像范围），主要的解决方案有：直接**忽略边缘**（这会使图像尺寸减小）；**补零**；**周期延拓**（假设图像是周期性的，在最上面一行上方复制最下面一行）；**镜像反射**（将最靠近边缘的像素向外反射一份）。

图像滤波任务中，按照处理的频域特性主要分为**低通滤波**（抑制高频分量，实现平滑）和**高通滤波**（增强高频，实现锐化），按照滤波运算的数学形式可以分为**线性滤波器**（表现为周围像素的加权和）和**非线性滤波器**（受周围像素具体值影响）。下面列举几类经典的滤波器：

- **均值滤波器**：

$$\frac{1}{9} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

一种**低通线性滤波器**，用于图像模糊与平滑处理。

- **高斯滤波器**：

$$\frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

属于**低通线性滤波器**，通过近似高斯分布实现更平滑的降噪效果。也可直接依据 $\mu = 0, \sigma = 1$ 的高斯函数生成更精确的权值矩阵。

- **中值滤波器**：一种**低通非线性滤波器**，在局部窗口中选取**中位数**作为输出，可有效去除“椒盐”噪声。
- **边缘检测滤波器**：属于**高通非线性滤波器**，主要分为两类：

1. 基于梯度的检测：计算梯度幅值

$$|\nabla a| = \sqrt{\left(\frac{\partial a}{\partial x}\right)^2 + \left(\frac{\partial a}{\partial y}\right)^2},$$

其中两个方向的梯度通常由 **Sobel 算子**给出：

$$\frac{\partial}{\partial x} : \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad \frac{\partial}{\partial y} : \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

在此基础上发展出的 **Canny 算子**是该类方法的集大成者，其典型流程为：**高斯降噪** → **Sobel 梯度计算** → **非极大值抑制**（若当前像素梯度幅值不是该方向上的**局部最大值**则置零）→ **双阈值检测**（设高阈值 T_H 与低阈值 T_L ，大于 T_H 的为强边缘，小于 T_L 的舍弃，中间值为弱边缘，仅保留与强边缘相连的部分）。该方法能在定位精度与抗噪性能之间取得良好平衡。

2. 基于二阶导数的检测：即拉普拉斯算子

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \Rightarrow \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix},$$

可同时检测图像中的边缘与纹理变化。

- **锐化滤波器**：属于高通线性或非线性滤波器。常见实现包括：

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix} - \frac{1}{9} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix},$$

或利用拉普拉斯增强公式

$$f' = f + \nabla^2 f,$$

将高频细节重新叠加以提升图像清晰度。

二维滤波器需要进行 n^2 次乘/加操作，效率相对低下，因此可以考虑将其分离为两个一维滤波器，例如高斯滤波可分离为 $\frac{1}{4} \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} \cdot \frac{1}{4} \begin{pmatrix} 1 & 2 & 1 \end{pmatrix} = uv^T$ ，于是 $a \otimes b = (a \otimes u) \otimes v^T$ ，成功将复杂度降至 $O(n)$ 。以上是**可分离滤波器**的情况，对于不可分离的滤波器，可以对其进行**奇异值分解**，化为 $H = \sum_i \lambda_i u_i v_i^T$ 形式再进行一维滤波。

线性滤波器最重要的性质是**线性平移不变性**，即 $h \otimes (f_0 + f_1) = h \otimes f_0 + h \otimes f_1$ ，并且 $g(i, j) = f(i+k, j+l) \iff (h \otimes g)(i, j) = (h \otimes f)(i+k, j+l)$.

(cv lec3 再完善一下)

2.3 图像修复/融合

图像的修复 (inpainting) 和融合 (blending) 都属于这样一类任务，其核心目标是在图像的某一区域内，根据周围信息生成自然、连续的结果。单纯的图像修复是相对简单的任务，只需要满足边界条件的前提下梯度尽可能平滑即可，也即

$$\operatorname{argmin}_f \iint_{\Omega} \|\nabla f\|^2, \quad s.t. f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

由变分法可求得其取最小值时即为 $\Delta f = 0$ ，这也因此被称为**拉普拉斯修复**。而在图像融合任务或是一些有约束的修复任务中，常常有一个**起指导作用的梯度场**（因为在图像融合任务中，色调可能需要改变，但梯度一般不变），此时任务变为使得输出图像梯度尽量接近梯度场，即

$$\operatorname{argmin}_f \iint_{\Omega} \|\nabla f - g\|^2, \quad s.t. f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

最小值条件相应地变为 $\Delta f = \text{div}(g)$, 这被称为泊松编辑.

在图像处理实际应用中, 只需要使用上述公式的离散形式即可 (假设图像间距为 1):

$$\frac{\partial^2 f}{\partial x^2} \approx \frac{\partial f_{i+0.5,j}}{\partial x} - \frac{\partial f_{i-0.5,j}}{\partial x} \approx f_{i+1,j} + f_{i-1,j} - 2f_{i,j} \Rightarrow \Delta f \approx f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1} - 4f_{i,j}$$

令 $\Delta f = 0$, 即得到迭代公式 $f_{ij} \rightarrow \frac{1}{4}(f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1})$, 若为泊松编辑, 则变为 $f_{ij} \rightarrow \frac{1}{4}(f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1} - \Delta F_{ij})$, 其中 $\Delta F_{ij} = F_{i+1,j} + F_{i-1,j} + F_{i,j+1} + F_{i,j-1} - 4F_{i,j}$ 即可。在实际运算中可以选择将步长减小以保持稳定: $f_{ij} \rightarrow f_{i,j} + \alpha(f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1} - \Delta F_{ij} - 4f_{i,j})$, $\alpha \in (0, 0.25]$.

2.4 图像拼接

由于手机的感受野有限, 无法有效得到全景图。本节就将讲解全景图是如何构建的。主要分为四个步骤: (使用 SIFT 特征等) 抽取特征-进行特征匹配-求得一个变换 (使得特征点对齐) -图像融合与过渡 (blend image)。前两步在此前的课程中已经涉及, 下面从求变换开始。(去除野点)

图像滤波主要是改变值域, 图像变换主要是改变定义域, 即 $g(x) = f(T(x))$, T 是一个有 6 个自由度的矩阵 $\begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}$ 。

如何求解? 写成大矩阵, 则每对点确定两行, 于是至少需要三个不共线的点。

$$E = \|At - b\|_2^2 = t^T A^T At - 2t^T A^T b + b^T b$$

令导数为 0, 得到最小值在

$$A^T At = A^T b$$

由于 A 的列秩为 6, 于是 $A^T A$ 通常是满秩的, 于是只需对 $6*6$ 矩阵 $A^T A$ 求逆, 即可得到:

$$t = (A^T A)^{-1} A^T b$$

即最小二乘解。

而对于射影变换, 有 8 个自由度 (why? 本来是 9 个自由度, 缩放消掉一个), 写成矩阵形式 (1) , 也即等式

还需加一个约束, 可以设定 $h_{22} = 1$ 或者 $[h_{00}, h_{01}, \dots, h_{22}]$ 向量的模长为 1. 同样地写成大矩阵形式, (1) , 于是只需要求解 $Ah = 0$ (在 $\|h\| = 1$ 的约束下), 于是可以写出 (基于拉格朗日乘子的) 能量函数

$$E = \|Ah\|^2 + \lambda(\|h\|^2 - 1) = h^T A^T Ah + \lambda h^T h - \lambda$$

求导得 $A^T A h = \lambda h$, 也就是转化为求特征值问题, 9 个特征值中怎么选取? 代入可以得到 $E = \lambda$, 于是取最小特征值对应的特征向量 h 即可。

回到 feature matching。处理野点是一个重要的问题。RANSAC 是解决这一问题的重要方法。其过程为: 重复 k 次【选取 s 个样本, s 为下一步求解变换需要的最小点数 (例如对于……就是 4 个点), 拟合一条直线, 数出 inliners 的个数】，选出最多 inliners 的直线, 再根据所有 inliners 调整直线 (最小二乘), 这种方法会失败当且仅当每次迭代选出的点里都存在野点, 根据计算这个概率较低 (这也是为什么每次不能取太多样本的原因), 因此成功率较高。

(vcl/10) 随机取两个点取到野点概率比较低。

最后我们来到了图像融合。之前讲了 Laplacian 金字塔来做融合, 其高频没问题, 但是低频部分仍然是加权平均, 无法处理颜色差别特别大的部分。

介绍 Poisson 编辑。核心思想是, 我们需要改变色调, 但是梯度不变。??? 大型稀疏矩阵。

2.5 图像压缩 (图像频域变换)

本节将从 **JPEG 有损压缩** 展开, 相对于 PNG 等无损压缩, JPEG 在编码过程中故意丢掉了很大一部分信息 (最终大小仅为原图像的 5%), 却保证解码后图像仍非常接近人眼感知。这一效果是如何做到的呢?

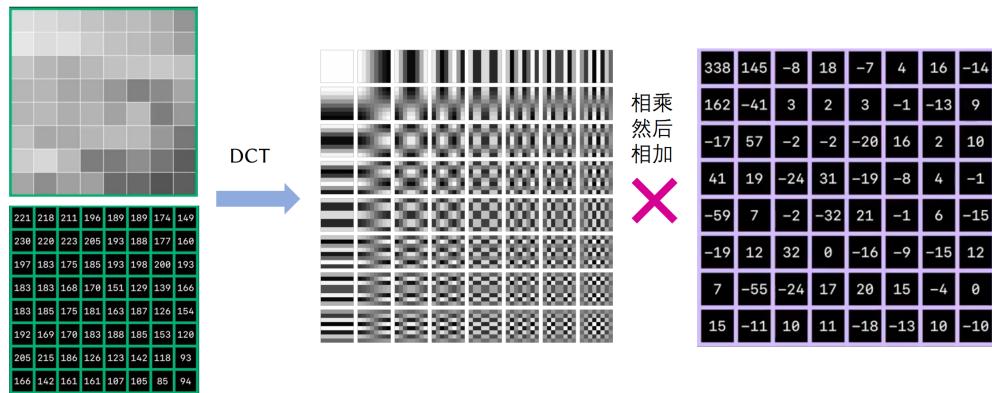
首先, 基于人类对亮度比对颜色更敏感的事实, 将 RGB 色彩空间重构为 YCbCr 空间 (亮度-蓝色色差-红色色差), 并保留 Y 通道不变, 对 Cb-Cr 通道分别进行平均降采样或取左上像素的子采样, 将图像压缩为原来的 50%, 但这离我们的目标还相距甚远, 我们需要对亮度通道进行一些操作。

前面也提到过, 人眼对高频信号的敏感度较低, 而拍摄所得图像中高频信号所占比例亦相对较少。若我们能够将这些既不主要 (对人眼而言) 又不重要 (在能量上较弱) 的高频成分一并舍弃, 便可以在几乎不影响感知质量的前提下实现对图像的高效压缩。问题因此转化为: 如何提取并编辑不同频率的信号。这就涉及到**图像的频域变换**, 即像素空间与频率空间之间的相互转换。

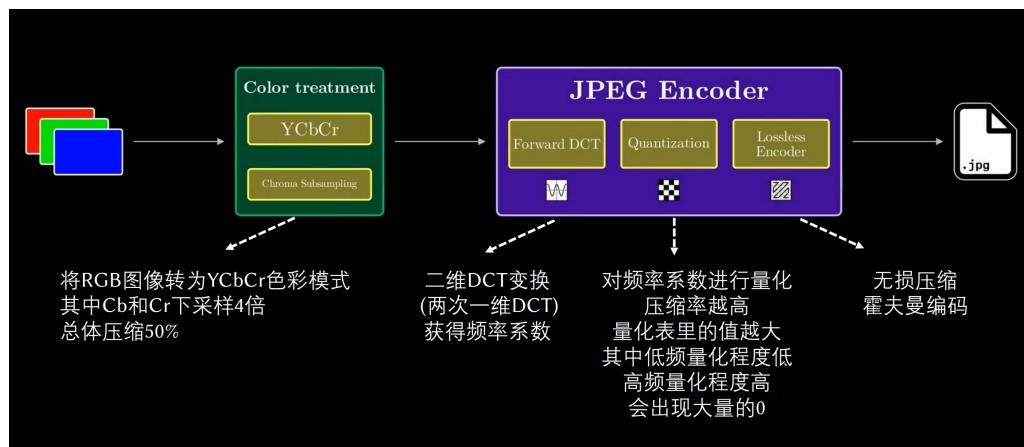
对于这一问题, 可以自然地回忆起傅里叶变换 (将函数分解为一系列三角基函数):

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos nx + b_n \sin nx)$$

在 JPEG 压缩中所采用的思想, 正是傅里叶变换的推广形式。设我们考虑一个 1×8 的一维 block, 由于采样点数有限, 傅里叶分解的频率分量必然存在上限 (即求和式从 $\sum_{n=1}^{\infty}$ 改为 $\sum_{n=1}^8$)。此外, 信号的定义域是有限的, 因此可以进行偶延拓, 将问题转化为更为简洁的余弦展开。当这一思想自然推广至二维空间时, 便得到 JPEG 压缩中至关重要的**离散余弦变换 (DCT)**。其具体形式如下图所示:



具体而言，我们把 Y 通道（实际上 Cb/Cr 也可以进行同样的操作）分解成 8×8 的 block，对于每个 block 作 DCT，得到频域表示，可以发现较大的数值通常集中在图像的左上，也就是低频区域，因此我们的目标便是将右下的高频信号尽量置为 0，采取的方法称为 **量化**，操作是定义一个 8×8 的量化表，在编码时将每个像素整除量化表内对应值，并在解码时重新乘回去。如此，通过人为设计量化表（例如，将左上的值设的较小，将右下的值设的较大），可以实现右下大面积置 0. 接下来，通过从左上到右下的 Z 字形采样可以得到一串末尾有大量 0 的序列，最后将连续的 0 合并进行 Huffman 编码（事实上这一过程更为复杂，但这里不做展开），即可实现压缩，总体流程如下图：



而在地震波/核弹爆炸检测等任务中，信号定义域并非有界，无法偶延拓，因此只能用**离散傅里叶变换 (DFT)**，进一步可以利用三角函数信号在采样点的重叠，采取分治策略将复杂度从 $O(N^2)$ 降至 $O(N \log N)$ ，这就是应用极广的**快速傅里叶变换 (FFT)**（当然，它也可以用作信号压缩）。

第三章 不那么基础的图像处理

3.1 图像插值与超分辨率重建

本节主要介绍单图像超分辨率 (SISR) 问题。与多图像超分辨率不同，后者可以依靠多张图像之间的信息冗余通过插值实现复原，而前者仅依赖单张低分辨率图像，因此是一个欠定问题（即其解不唯一）。为获得合理的高分辨率重建结果，必须引入先验知识进行约束，这也引出了多种基于深度学习的重建算法。

在深入这些算法之前，需要先梳理不依赖先验的基本上采样方法——**图像插值**。按照空间连续性从低到高，可分为以下几种：

- **最近邻插值**：顾名思义，易出现块状伪影。
- **双线性插值**：综合考虑邻域四个像素的加权平均，结果更平滑。
- **双三次插值**：三次插值是基于两端点的函数值及其导数值拟合三次函数，双三次插值是其在二维空间中的自然推广。其效果通常优于双线性插值，但可能出现振铃现象，即边缘区域出现突然变亮或变暗的伪影。

但无论如何插值，都无法创造我们想要的细节，由此引出一些基于学习的方法。

首先介绍图像的**稀疏表示**。参照 DCT 的思想，我们希望用若干基底表示图像，但与 DCT 不同，稀疏表示显著增加了基底的数量，并且通过**数据驱动**的方式从样本集中学习这些基底，这样的基底集合称为**过完备字典**（一个列数远大于行数的矩阵）。在这种字典中，每个图像块都可以表示为若干列向量的**稀疏线性组合**，即仅有极少数非零元素。形式上可写为： $X = D\alpha$.

将其进行数学建模，我们即是要优化如下式子：

$$D = \arg \min_{D, \alpha} \|X - D\alpha\|_2^2 + \lambda \|\alpha\|_0$$

由于零范数（非零元素的个数）难以优化，实际应用中会用一范数代替。同时，在实际学习中，为了防止模型“走捷径”（ $\|\alpha_i\|$ 很小， $\|D_i\|$ 很大），还需要添加额外约束，于是得到最后的优化公式：

$$D = \arg \min_{D, \alpha} \|X - D\alpha\|_2^2 + \lambda \|\alpha\|_1 \quad s.t. \|D_i\|_2^2 \leq 1$$

由于同时优化两个值是困难的，于是我们对 D, α 进行交替最小化，如此转化为得两个规划问题都是可解的。

具体利用在超分辨率重建中，在训练过程中同时构建低分辨率字典和高分辨率字典：

$$D_h, D_l = \arg \min_{D_h, D_l, \alpha} \|X - D_h \alpha\|_2^2 + \|Y - D_l \alpha\|_2^2 + \lambda \|\alpha\|_1$$

在测试过程中输入低分辨率图像 y ，根据其优化稀疏编码 α ，进而计算出高分辨率图像块 $x = D_h \alpha$ 。这种方法能够有效补全细节，但其缺点是测试阶段计算开销较大，且训练过程耗时。

为了解决构建字典的效率问题，**邻域嵌入**不再显式构建全局字典，而是直接在**训练样本空间**中进行局部匹配：对于输入的低分辨率图像块，首先在训练集中寻找与之**最相似的** K 个低分辨率样本块，然后仅利用这些邻域样本计算线性组合系数，从而实现局部的稀疏重建。其优化目标为：

$$\alpha_i = \arg \min_{\alpha_i} \|x_l^i - N_l^i \alpha_i\|_2^2 \quad s.t. \mathbf{1}^T \alpha_i = 1$$

当然在训练集中找相似块的开销依旧不小，并且高分辨率图像块的加权平均也会使输出模糊，因此还需要更先进的算法。

进入深度学习时代后，**SRCNN** 成为深度图像超分辨率方法的开端。其结构十分朴素：在训练阶段，使用高、低分辨率图像对作为数据集，先通过**双三次插值**等传统方法对低分辨率图像进行上采样以实现**尺寸对齐**，再将上采样后的图像输入由三层卷积层组成的网络中，最小化输出结果与真实高分辨率图像之间的**L2 损失**即可（测试阶段流程相同）。该方法实现了端到端的超分辨率重建，但由于所有卷积操作均在高分辨率空间进行，效率较低。

为此，**Fast-SRCNN** 在效率上进行了两项关键改进：

- **延后上采样操作**：将上采样从输入阶段移至输出阶段，使用**转置卷积层**实现图像空间分辨率的提升。这样不仅省去了插值过程，还将中间卷积操作从高分辨率空间移至低分辨率空间，从而显著提升计算效率。
- **引入降维与升维结构**：为减少特征通道数带来的计算负担，**FSRCNN** 在输入后加入**降维层**（使用 1×1 卷积核，仅在通道维度上进行线性组合以实现降维），并在转置卷积层前设置**升维层**以恢复特征维度。通过这种“降维—映射—升维”的结构，网络在保持表达能力的同时大幅降低了计算量。

得益于上述改进，**FSRCNN** 可以在更小的计算代价下使用更深的卷积层结构，从而获得更优的重建效果。

但其中的转置卷积层会造成**棋盘格伪影**（因为步长不能整除卷积核大小，会造成部分区域的重叠），因此我们需要找到更好的上采样方法。可以简单改为双线性插值 + 卷积，也可以使用**亚像素卷积**，即将 $(H, W, r^2 C)$ 重排为 (rH, rW, c) ，后者在超分辨率任务中更常用。

此外，损失函数的设计也可以进一步改进。传统的**L2 损失**虽然易于优化，但倾向于生成过于平滑的结果——网络会输出所有可能高分辨率图像的平均值，从而导致纹理细节模糊。此类模型的

PSNR (可视作图像相似度指标) 虽高, 却与人类视觉感知并不一致。因此, 可引入**感知损失**, 即不再在像素空间中计算误差, 而是在**高层语义特征空间**中衡量生成图像与真实图像之间的差异 (更直观地说, 就是对网络的中间卷积层输出计算 L2 损失), 这显著提高了人类感知下的图像质量。

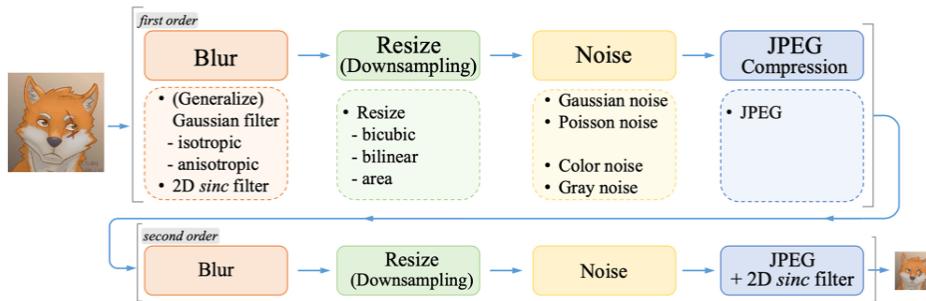
我们同样可以将 GAN 的思想融入其中, 得出 **SR-GAN**, 其中判别器的任务是**鉴别图像的真伪**, 这事实上是更高层的指标, 因而其同样 PSNR 不一定高, 但看上去更真实。其损失函数为:

$$l^{SR} = l_X^{SR} (\text{L2 损失或感知损失, 引导输出“像原图”}) + 10^{-3} l_{Gen}^{SR} (\text{对抗损失, 引导输出“像真的”})$$

ESRGAN (Enhanced SRGAN) 则在多维度上对其做出了改进:

- 移除 Resnet 结构中的 BN 层, 防止色彩对比度的归一丢失, 也防止训练集和测试集**分布显著不同**导致重建出的图像相邻帧色调抖动, 产生伪影。
- 引入 RRDB 层, 即密集连接的多层次残差嵌套, 可以有效防止信息丢失。
- 将对抗损失部分改为**相对对抗损失**, 即判断“一张图像是否比另一张图像更真实”。

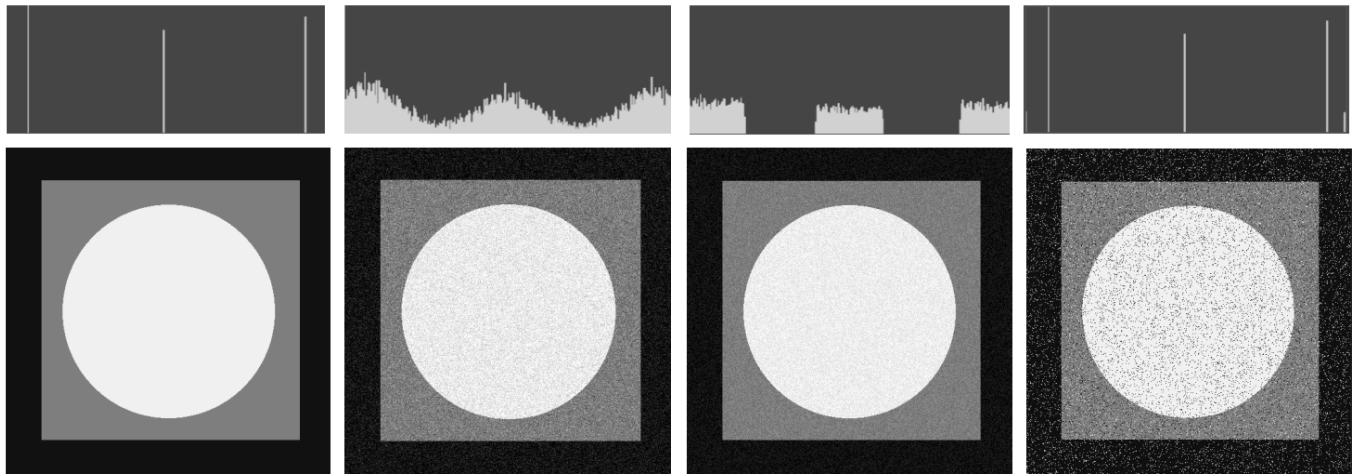
这在 ImageNet 等测试集上表现出众, 但却对**真实世界的图像重建**表现乏力。原因是在以上的所有模型中, 我们都先验地假设了**下采样的降质模式**, 但真实情况中我们并不知道是由何种方式下采样 (也即, 对于下采图像 $y = Hx + v$, 我们实际不知道降质矩阵 H 和高斯白噪声 v)。对此的解决方案是**人工造数据**, 模拟大量可能的降质方案, 让网络能处理多种方案。具体方法如下图 (模糊、下采样、加噪、JPEG 压缩):



从而得到降质图像进行训练。这便是 **Real-ESRGAN** 的核心优化方向。还有一些其他层面的小优化, 包括使用 Pixel Unshuffle 统一输入尺寸从而实现不同倍率的重建; 判别器改用 U-Net, 并判断每个像素的真假; 先不加 GAN 得到平滑的初始输出, 再加入 GAN 进一步训练等。

3.2 图像去噪

在上一节中我们提出了图像降质模型 $y = Hx + n$, 本节我们就重点关注对于噪声项 n 的建模与抑制。常见的图像噪声类型包括**高斯噪声** (最常见)、**均匀噪声**以及**椒盐噪声**。它们的视觉效果与灰度直方图分布如下图所示:



用普通的低通滤波器固然可以去噪，但图像的边缘也同时被模糊了。为了保护边缘，我们提出了**双边滤波**的思想，选择对空间距离较近且像素值相似的像素点进行平滑，具体而言其使用两个高斯分布进行加和：

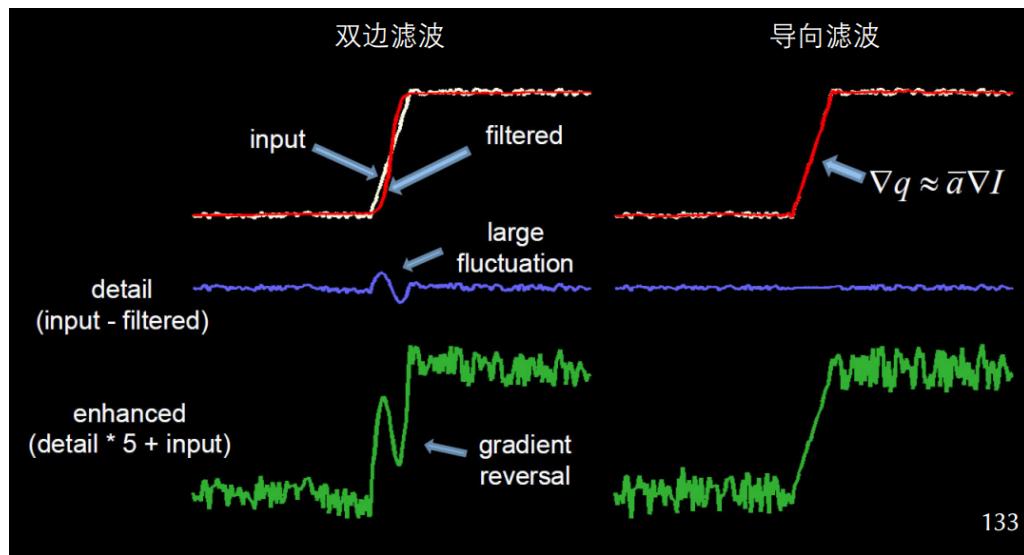
$$w(i, j, k, l) = \exp \left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_s^2} - \frac{\|\mathbf{f}(i, j) - \mathbf{f}(k, l)\|^2}{2\sigma_r^2} \right)$$

这一方法有一个经典的拓展：**联合双边滤波**，其使用另一张**引导图像**来判断像素值的相似性。举一个具体的例子，在低光环境下，不开闪光灯拍出的照片（A）通常噪声较多，而开了闪光灯后的照片（B）更为干净，但光照条件与实际情况高度不一致，此时可以使用 B 作为引导图像，通过 B 中的像素值相似性更新 A 中的像素信息，也即

$$I'_A(k, l) = \frac{1}{W_{k,l}} \sum_{i,j} \exp \left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_s^2} - \frac{\|\mathbf{f}_B(i, j) - \mathbf{f}_B(k, l)\|^2}{2\sigma_r^2} \right) I_A(i, j)$$

然而，双边滤波仍然保留了空间距离的权重。在去噪任务中，这一特性影响较小；但在**细节增强**中，边缘区域的像素仍会受到相邻外部像素的影响，导致滤波结果在边缘处的像素值高于区域内部。随后在细节增强过程中（即原图减去 k 倍滤波层），这种不对称的偏移会被放大，从而产生异常的过亮或过暗现象，也即所谓的**梯度反转**。

为了解决这一问题，提出了**导向滤波**，其利用一张**引导图像**引导图像的输出的梯度结构，具体而言，它通过在局部窗口内拟合线性关系 $q_i = aI_i + b$ ，从而使输出图像的梯度与引导图像的梯度保持线性相关，即 $\nabla q = a\nabla I$ ，这与双边滤波的区别可以从下图中看出。



于是优化目标即为：

$$\min_{a,b} \sum_i (aI_i + b - p_i)^2 + \epsilon a^2$$

其中 ϵa^2 为正则化项，防止局部过于平坦时 a 数值过大。令 $\frac{\partial L}{\partial a} = \frac{\partial L}{\partial b} = 0$ ，可以求出上述优化问题的解析解：

$$\begin{cases} a = \frac{\text{cov}(I,p)}{\text{var}(I)+\epsilon} \\ b = \bar{p} - a\bar{I} \end{cases}$$

在实际图像计算中，我们在图像上滑动多个局部窗口，分别求解每个窗口对应的线性系数 a_k 和 b_k 。随后，对于任意像素点，将其所在的所有窗口的输出结果 $q_k = a_k I_k + b_k$ 进行平均，即可得到该像素的最终滤波值。

我们指出，这样的结果能够实现**边缘保持**的效果：在平坦区域 ($\text{var}(I) < \epsilon$) 中， $a \approx 0$, $b = \bar{p}$ ，输出相当于对输入进行平滑去噪；而在边缘区域 ($\text{var}(I) > \epsilon$) 时， $a \approx \frac{\text{cov}(I,p)}{\text{var}(I)} \neq 0$ ，输出的梯度能够随引导图像变化，从而有效地保持边缘结构。

当输入和引导图像是同一副图像，也即 $I = p$ 时，该算法成为一个边缘保持滤波器，此时 $a = \frac{\text{var}(I)}{\text{var}(I)+\epsilon}$, $b = (1-a)\bar{I}$.

接下来介绍的两种算法在核心思想上与前述滤波方法有所不同：它们不再依赖于空间邻域的加权平均来去除噪声，而是基于对自然图像统计特性的**先验认知**。这一认知认为，噪声与纹理的区别并非取决于像素之间的局部距离，而在于图像纹理具有显著的**自相似性**——即在全图范围内，许多局部结构在不同位置重复出现。

基于这一思想，**非局部均值滤波**提出了一种更具全局性的去噪方式。与双边滤波仅在局部区域内比较像素灰度不同，其认为仅比较单个像素无法准确反映其结构相似性。相反，它在较大的**非局部搜索区域**内，寻找与目标像素所在**图像块**相似的区域，并根据这些图像块之间的相似程度计算滤波权重。具体而言，算法首先设定一个较大的搜索窗口，以目标像素为中心，在该窗口内滑动图像块，通过测量每个候选图像块与目标图像块的相似度来确定权重，最终利用这些权重对像素值进行

加权平均：

$$I'_p = \frac{1}{\sum G_{\sigma,\theta}(\|P(p) - P(q)\|)} \sum_{P(q) \in R_p} G_{\sigma,\theta}(\|P(p) - P(q)\|) I_q$$

其中权重为 (θ 为噪声的标准差, 低于该标准差的认为是同一个块, 权重置为 1)

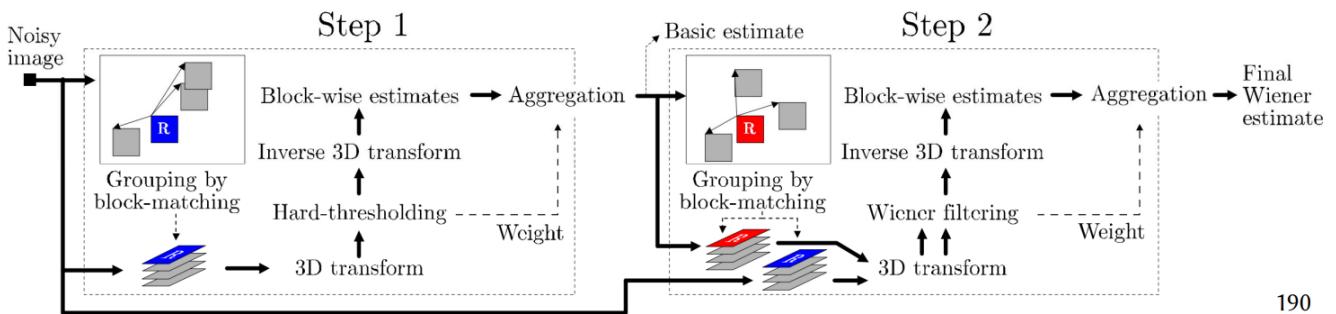
$$G_{\sigma,\theta}(\|P(p) - P(q)\|) = \exp\left(-\frac{\max(\|P(p) - P(q)\|_2^2 - 2\theta^2, 0)}{2\sigma^2}\right)$$

最后一种传统算法——**BM3D (Block-Matching and 3D Filtering)**, 同样以匹配相似块为核心思想, 但其关键改进在于将滤波过程从空域转移到频域。这种转换利用了频域中信号能量集中、噪声能量分散的特性, 使得信号与噪声在分布上更易区分, 从而实现了更高精度的去噪效果。

此外, BM3D 的另一重要设计是在三维频域空间中进行滤波。具体而言, 它将二维图像中的相似块堆叠成一个三维数据体, 并在三维空间中执行变换与阈值操作。之所以采用这种方式, 是因为在二维空间中直接对较小的频域系数进行截断, 在高噪声情况下可能误杀低频信号。而将相似块堆叠至三维后, 可以显著增强结构一致性, 使有效信息在频域中更加稀疏集中, 从而实现更稳健的去噪。

在频域中完成滤波操作后, 将结果通过逆变换回到图像域, 并将其贴回原图对应位置。对于不同块间的重叠区域, 采用加权平均以保证平滑过渡, 从而获得完整的去噪图像。

在实际实现中, BM3D 的去噪过程通常分为两个阶段: 首先, 对原始图像 I 进行**阈值滤波**, 将变换域中幅值小于阈值的系数置零, 以去除明显的噪声成分, 得到初步去噪结果 I' 。随后, 以 I' 作为**参考图像**, 对原图再次执行维纳滤波。在该过程中, I' 中幅值较小的位置对应较低的维纳滤波权重, 从而在保留主要结构信息的同时, 有效抑制低频噪声, 实现更精确的去噪效果。具体效果见下图:



190

该方法同样适用于**彩色图像去噪**: 首先将图像从 RGB 空间转换至 YCbCr 空间, 在亮度通道 (Y 通道) 中执行块匹配以确定相似块的位置, 随后在三个通道上分别独立进行去噪处理, 即可实现彩色图像的协同滤波。

深度学习方法在图像去噪任务中同样取得了显著成果。其中最早且具有代表性的是 **DnCNN**, 其结构并不复杂, 采用典型的卷积神经网络堆叠形式: Conv + ReLU + $k \times$ [Conv + BN + ReLU] + Conv。该模型的关键设计在于, 网络并非直接学习去噪后的图像, 而是学习带噪图像与真实图

像之间的残差。这种残差学习的策略实质上将“去噪”转化为“去除原图信号”的过程，使学习过程与噪声种类关系不大，从而对不同类型的降质具有更好的鲁棒性与泛化能力。

然而，以 DnCNN 为代表的早期方法由于未能准确建模真实噪声，在实际拍摄图像上的表现并不理想。针对这一问题，CBDNet 提出了多项关键改进。首先，在噪声建模上，CBDNet 将噪声方差视为与相机接收到的辐照度相关的变量 σ_s ，而非简单的加性白噪声模型；同时，还考虑了相机成像过程和 JPEG 压缩对噪声分布的影响，使合成噪声更贴近真实成像环境。

其次，在网络结构上，CBDNet 采用了盲去噪框架，由两个子网络组成：一个噪声估计子网络用于预测噪声水平图，另一个非盲去噪子网络（由五层全卷积结构构成）在噪声估计的指导下完成图像恢复。

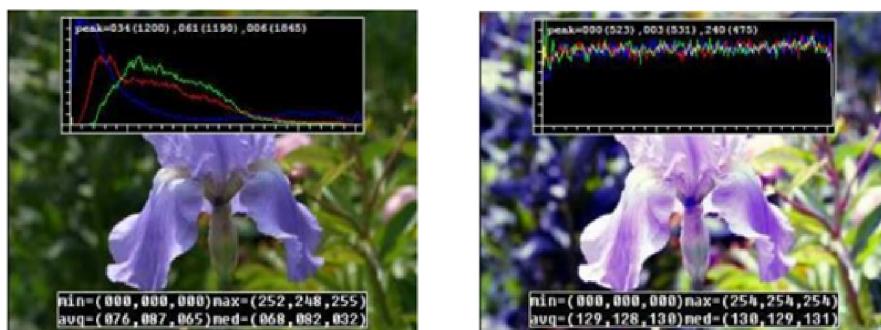
在损失函数设计上，CBDNet 综合了三类约束：重建损失（保证去噪图像的保真度）、总变分 (TV) 损失（约束噪声估计图 $\sigma(y)$ 的平滑性，使得其梯度尽量小）以及非对称损失。其中，非对称损失的动机在于人眼更容易接受过度去噪而“略微模糊的图像”而非欠去噪而“仍含噪声的图像”，因此当网络低估噪声水平时会施加更高的惩罚权重。

最后，在训练策略上，CBDNet 结合了合成数据与真实数据进行联合训练：在合成数据上使用全部损失项，而在真实数据上仅采用重建损失与 TV 损失，舍弃非对称损失。两类数据交替训练，使模型具备对真实噪声的强泛化能力。

3.3 图像直方图与低光照增强

本节聚焦于另一种降质形式：即在图像采集前的低光。我们希望进行低光照增强，但简单地增强 L 通道适应性太差，容易使图像白成一片，需要根据图像特性设计提亮方法。

一种重要的传统方法是直方图均衡化。该方法利用像素各通道值的累计分布函数重新映射原图的 RGB 像素，使输出分布更加均匀，从而增强图像对比度。下图展示了均衡化前后的效果对比：

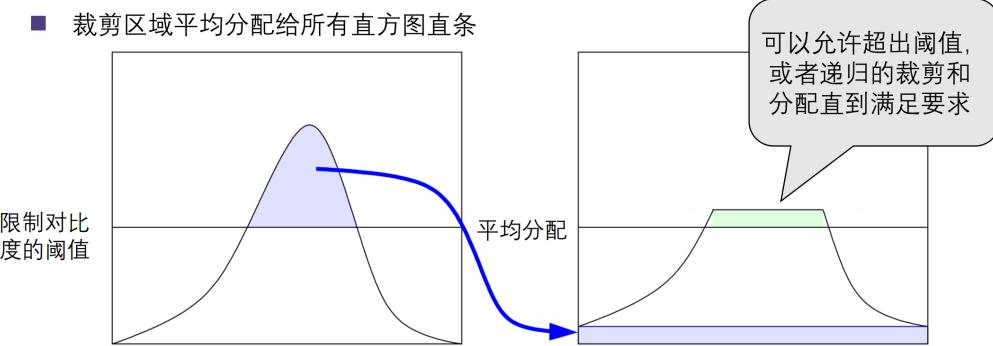


其实现原理是（以灰度图为例）：将概率分布（排名）处于 $n\%$ 的像素点映射为灰度值 $255 \times n\%$ ，从而得到趋近均匀的输出分布。对于彩色图像，若直接对 RGB 通道分别均衡化会产生明显色偏，通常的解决方案是转换到 HSV/HSL 空间，仅对 V/L 通道进行处理。

然而，全局直方图均衡化可能使灰度集中区域的对比度被过度拉伸，甚至放大噪声；同时，若

图像整体偏暗（或偏亮），亮部（或暗部）会被挤压到极值处，导致细节丢失。为缓解这些问题，提出了**限制对比度自适应直方图均衡化（CLAHE）**，其核心思想如下：

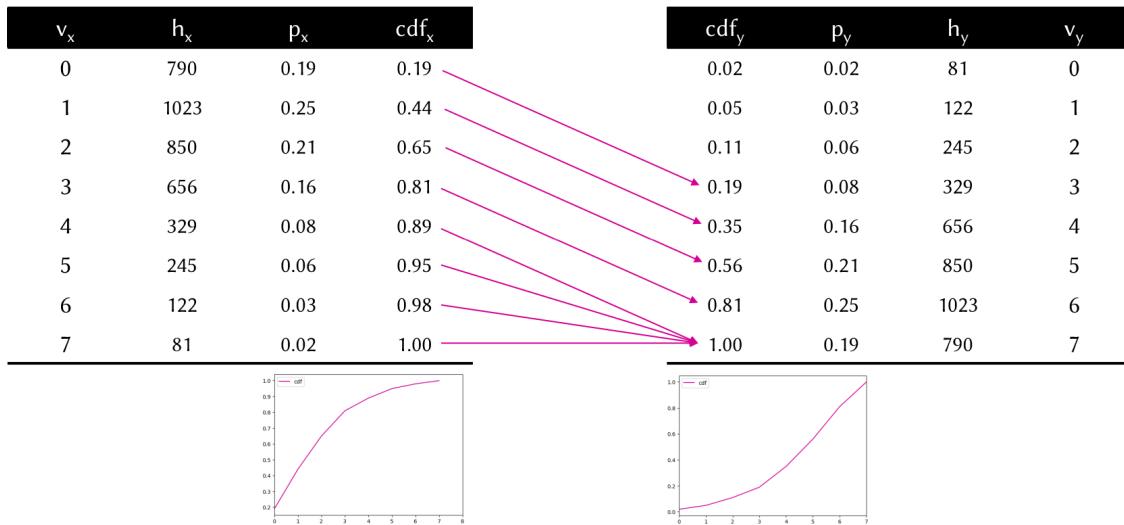
为了避免局部对比过度增强，CLAHE 会对直方图中过于集中的部分进行裁剪，并将裁剪后的像素概率均匀分配到其他灰度段，如下图所示：



同时，为避免图像中局部区域明显亮于或暗于整体，CLAHE 采用自适应策略：将图像分块，对每个块的**中心像素**仅使用该块的直方图计算变换函数，对于其他像素的变换函数则通过**双线性插值**得到（角落像素直接使用所在块的变换函数）。分块能够减少局部过曝/欠曝，但若分得过细会使灰度过度集中、对比度增强过度并引入噪声。因此，对每个块进一步应用限制对比度便形成了**CLAHE**。该方法在低光照增强等任务中表现良好，其效果如下图所示：



以上算法的目标是将直方图的累积分布函数拉伸为近似直线。而更一般地，我们可以借助另一张**引导图像**的直方图作为参考，使待处理图像的灰度分布向其靠拢，这便是**直方图匹配**。其实现如下图一目了然：



然而，基于直方图的处理方法普遍缺乏空间上的适应性，因此需要引入更智能的低光照增强算法。下面首先介绍一种经典的去雾方法——基于 **dark channel** 的图像去雾——图像去雾事实上和低光照增强是相似的任务。

该方法提出，雾天成像过程可被建模为： $I = J \cdot t + A \cdot (1 - t)$ ，其中 I 、 J 分别代表有雾图像与真实场景，无雾图像； A 为全局大气光； t 为传输率。我们的目标是从 I 中重建出 J, A, t 。

这一模型显然是欠定的，但作者巧妙地引入了一个重要的先验：自然图像的暗通道（对图像取 $\min(R, G, B)$ 后再进行最小值滤波）通常高度集中在 $[0, 16]$ 范围内（原因在于场景中广泛存在阴影，以及物体颜色饱和导致 RGB 通道中总会有某个分量较小）；而在有雾图像中，由于空气光的影响，暗通道显著变亮。由此即可估计 A ：**取暗通道中数值最大的区域对应的像素作为大气光**。

接下来，原模型可写为

$$D\left(\frac{I}{A}\right) = D\left(\frac{J}{A}\right) \cdot t + 1 - t \approx 1 - t,$$

从而得到 $t = 1 - D(I/A)$ ，进而即可恢复出 J 。但由于暗通道依赖最小值滤波，估计得到的 t 往往呈现块状伪影，此时可利用上一节介绍的**导向滤波**，以有雾图像作为引导，对 t 进行边缘保持平滑，以获得更精细的传输率估计。

这一方法在图像去雾任务上取得了不可思议的成功，但其针对暗通道的天才先验事实上也一定程度上限制了其推广到低亮度增强的能力（“亮通道”似乎没有那么良好的性质）。

进入深度学习时期，人们同样设计出了效果更为理想的低光照增强算法。

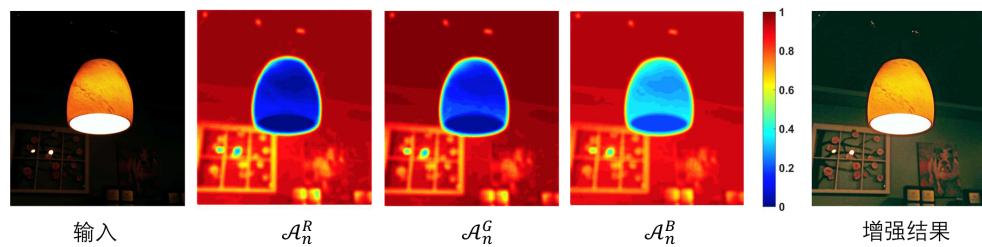
Retinex-Net 基于传统的 Retinex 理论，即人眼感知的图像可分解为**反射分量**（物体固有属性）和**光照分量**（场景光照强度）。因此，低光照增强的关键在于调整光照分量。然而传统方法在复杂光照环境下往往难以准确分离光照，且提升效果有限。为此，Retinex-Net 分别对分解与增强过程设计网络进行学习：

在分解阶段，通过训练 **Decom-Net**，对一组低光照/正常光照图像联合训练，并施加图像重建约束、反射率一致性约束（反射分量相近）以及光照平滑约束，从而学习得到光照分量；随后将低光图像 I_{low} 输入 **Enhance-Net**，得到增强结果 \hat{I}_{low} 。

这种方法能够产生较好的光照增强效果，但仍存在图像真实性不足等问题，并且高度依赖成对的高/低光照数据，获取成本较高。

Enhance-GAN 引入对抗训练策略，从而减轻了对配对数据的依赖。通过训练一套编码器—解码器结构作为生成器，再利用判别器以真实高光照图像为参照进行真假判别（文中采用全局与局部双判别器，分别评估整体与细节的合理性），即可实现低光照增强。

Zero-DCE 则进一步摆脱了对高光照数据的需求，直接从低光照图像中学习**提亮曲线**以生成增强结果，并以若干自然图像先验作为训练目标。具体而言，采用 DCE-Net 学习曲线参数 $\mathcal{A}_n(x)$ ，迭代构建提亮曲线 $LE_n(x)$ ，其效果如下图所示：



其损失函数包括：空间一致性损失（邻域像素保持一致）；曝光控制损失（控制亮度偏离程度）；颜色恒常性损失（RGB 均值保持平衡）；光照平滑损失（TV 正则项）。消融实验表明上述各项均不可或缺。该方法实现了无需参考图像的自监督训练范式，具有很强的自适应性。

本节所有关于低光照增强的算法一览如下图：

图像直方图和低光照增强		
	基于曲线调整	基于Retinex理论
传统方法	手动PS曲线调整 适合经验丰富者 直方图均衡化 图像统计数据匹配	NPE, LIME, SPIE 基于图像先验 优化求解病态问题
深度学习	Zero-DCE 无需参考图像 图像自适应，高阶，逐像素的提亮曲线 轻量级网络	Retinex-Net 依靠配对的数据 结合理论与深度学习
		EnlightenGAN (纯数据驱动) 无需搜集配对的数据 单纯依靠数据驱动

第四章 三维重建

Input Image-Camera Calibration- Correspondences- Depth Maps-Depth map fusion- 3d reconstruction

单视角奇异性。

相机配准。 n 个参考点。求内参 + 外参的大矩阵：12 个参数，需要六个参考点，与此前的 Homography 相似，写成大矩阵，使用拉格朗日乘子法得到 $E = \|Ap\|^2 + \lambda(\|h\|^2 - 1)$ ，参照此前的推导可得 p 即为最小特征值对应的特征向量。

得到了 $P = \begin{pmatrix} KR & Kt \end{pmatrix}$ ，考虑 KR 这个矩阵，其中 K 是上三角矩阵， R 是正交矩阵，会想到线性代数中的 QR 分解，实际上现在我们需要做 RQ 分解，是从最后一行开始做正交化（而非 QR 分解中从第一列开始）

(RQ 分解不够精确，为了将代数误差转化为真实的投影误差) 在实际过程中，如果想做的更好，可以考虑用非线性方法。 $\sum_i \|\text{proj}\|$ 牛顿法/梯度下降法。缺点是需要一个好的初始。

通过刚刚的操作，我们成功估出了相机的参数。现在的问题是，如果相机参数已经估出来了，也可以使用 sift feature 作 correspondence，是不是可以将 3D 坐标算出来。Triangulation (三角化) 是通过多个视角的图像点来估计该点在三维空间中的位置。

集合的方法，做公垂线，找公垂线段的中点。实际应用中还是希望把这个优化出来，也就是找到 X ，使得最小化 $\|\text{proj}(P_1X) - x_1\|^2 + \|\text{proj}(P_2X) - x_2\|^2$ 。这种方法非线性不够简单。

我们讨论线性优化。于上面的……相似，实际上就是 $x_1 \simeq P_1X, x_2 \simeq P_2X$ ，也可以写成叉乘形式： $x_1 \times P_1X = 0, x_2 \times P_2X = 0$ 。实际上，茶城也可以写成矩阵的形式：

$$a = b$$

4 个等式 3 个未知量，就可以使用最小二乘法。

惯用的做法其实可以用四维坐标，更稳定。

现在其实已经可以造一个板子做三维重建了。后续讲的有些是对这一点的优化。

相机标定的第二模块——用灭点标定。适合于建筑物、正方体。

齐次坐标可以统一地表达无穷远点。 $X_\infty = \begin{pmatrix} D_1 \\ D_2 \\ D_3 \\ 0 \end{pmatrix}$

定义直角点为原点，11个自由度， p_1, p_2, p_3 各自等于消失点坐标，需要求出缩放因子。无法组合成一个矩阵。而 p_4 代表世界坐标的原点的投影点。所以 $v_1 \simeq p_1, \dots, 0$ ，由于齐次坐标，实际上只有8个等式，无法约束住11个自由度。

如何约束呢？三个方向的正交性！由于 $v_i \simeq P \begin{pmatrix} e_i \\ 0 \end{pmatrix}$ ，于是 $v_i \simeq KRe_i$ ，进而得到 $e_i \simeq R^T K^{-1} v_i$ ，利用正交性可得 $e_i^T e_j = 0$ ，从而得到约束 $v_i^T K^{-T} K^{-1} v_j = 0$ ，虽然约束非线性，但并不难求解。

$$\text{实际上, } K = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix}, \quad K^{-1} = \left(\frac{1}{f} \quad \dots \right)$$

At least two finite vanishing points are needed to solve.

求出 K 后，代入……可得 $r_i \simeq K^{-1} v_i$

优势：不需要2D-3D匹配，可以做成自动的/劣势：只适用于特定的场景，消失点在一些时候比较难找且误差比较大。

4.1 极线几何

双目作 calibration，能不能为 correspondence 提供帮助？极线几何就是提供这种的帮助。

原点相连称为基线（baseline），baseline 与平面交点（或者说另一个相机原点在该相机平面上的投影点）称为极点。再加上给定的物体点，形成的三角形称为极面，极面与相机平面又有两条交线 Epipolar Line（极线）

相交；平行（极点在无穷远处）；三点共线，极线退化。

投影点到底是哪个点呢（见图）重要的结论：如果我想找 correspondence，只需要在极线上找对应点就行了，大大减小了搜索空间。

实际上，由于噪音和不精确的配准，事实上不一定特别精确。

下面是数学推导，假设内参已知且世界坐标落在第一个相机的坐标系上。则估计矩阵为 $K \begin{pmatrix} I & 0 \end{pmatrix}$ 和 $K' \begin{pmatrix} R & t \end{pmatrix}$ 。

为了进一步简化，可以将内参矩阵乘到等式左侧，即 $x = K^{-1} x_{pixel} \simeq \begin{pmatrix} I & 0 \end{pmatrix} X$ ，得到标准化的图像坐标。同理， $x' = K'^{-1} x'_{pixel} \simeq \begin{pmatrix} R & t \end{pmatrix} X$ ，也即 $x' \simeq Rx + t$ ，于是 x', Rx, t 线性相关，于是其三重积 $x' \cdot [t \times (Rx)] = 0$ ，将其简化成矩阵的形式： $x'^T [t \times] Rx = 0$ ，即

$$x'^T E x = 0$$

这其中的 $E_{3 \times 3}$ 就被称为**本征矩阵（Essential Matrix）**，描述了两个相机之间相对的旋转和平移。其中 $l' = Ex = (a, b, c)^T$ 即为 x' 对应的平面上极线方程的三个系数 $(ax + by + c = 0)$ ，同理， $l = E^T x'$ 对应另一条极线。而极点在 E 的零空间里 $(\forall e', e'^T (Ex) = 0)$ 。

下面分析 E 的性质。由于 $[t \times]$ 秩为 2（可以验证行列式为 0），而 R 秩为 3，因此 E 秩为 2；由于旋转和平移各有 3 个自由度，又由缩放歧义性小调一个自由度，因此共有 5 个自由度。

再将内参矩阵代入得到 $x'^T_{pixel} F x_{pixel} = 0$, 其中 $F = K'^{-1} E K^{-1}$ 称为基础矩阵 (**Fundamental Matrix**) . 同样地, 极线可以表示为 $l' = F x_{pixel}, l = F^T x'_{pixel}$. 极点同样在零空间中。F 同样秩为 2. 旋转平移内参共 9 个自由度, 其不满秩和缩放奇异性分别消掉一个自由度, 因此共有 7 个自由度。

下面谈谈如何估计基础矩阵。可以通过 SIFT 特征等找到一系列匹配点 $x = (x, y, 1)^T, x' = (x', y', 1)^T$ (通过 RANSAC 算法消除野点) . 然后写成大矩阵, $\arg \min_{\|f\|=1} \|Uf\|^2$, 但没有解决不

满秩的约束。这里需要用到 SVD 分解。将其分解为 $M = U\Sigma V$, 然后作低秩估计, 将

$$\begin{pmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{pmatrix}$$

转化为

$$\begin{pmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$
, 即得到矩阵。这被称为八点算法 (低秩估计前有 8 个自由度)。实际运用中

需要进行一定的校正: 由于 $(x'x \quad x'y)$ 中的元素数量级差距较大, 容易造成数值运算的不稳定, 所以要对坐标进行归一化 (将坐标先缩放到 $[-1, 1]$, 八点算法计算完, 再将上一步的缩放合并进基础矩阵中: $F_0 = T^T F T$)

刚刚的方法相当于约束了代数误差, 同样也有非线性的方法来约束几何误差: 先用代数方法求出比较好的 F, 再通过梯度下降优化 $L = \sum_i \dots \dots$

求出 F 后, 若有内参矩阵, 则就有了 E, 否则就只有一个弱的参数估计。求出了 E 之后, 经过一些复杂的推导, 也可以求出 R 和 t。

4.2 双目立体视觉

核心任务: 输入: 一对已知相机矩阵 (已标定) 的立体图像。

输出: 稠密深度图 (dense depth map)

.....

简单的方案: 两台相机的成像平面彼此平行, 并且也平行于基线 (baseline)。

基线是连接两台相机光心的那条线。两台相机的光心在同一水平高度 (即没有垂直偏移)。两台相机的焦距相同。

这些条件共同保证了: 同一三维点在两幅图像中的对应点位于同一水平线上。此时退化成扫描线。扫描线后再作 Triangulation, 得到深度图。

但现实并不总是那么美好。相机平面通常不平行。因此需要进行立体校正 (**Stereo Rectification**)。其核心步骤是: 调整相机方向 (对齐朝向, 使用八点算法), 旋转使平面与基线平行 (需要介绍!), 缩放校正畸变。

事实上, 三角.....也可以优化。..... (推导) 有用的结论: $z = \frac{fB}{x - x'}$, 其中就是需要求 $x - x'$ (视差) 能看出 B 越大, 误差越小。

难点反而来到了 Examine all pixels on the scan line and pick the best match

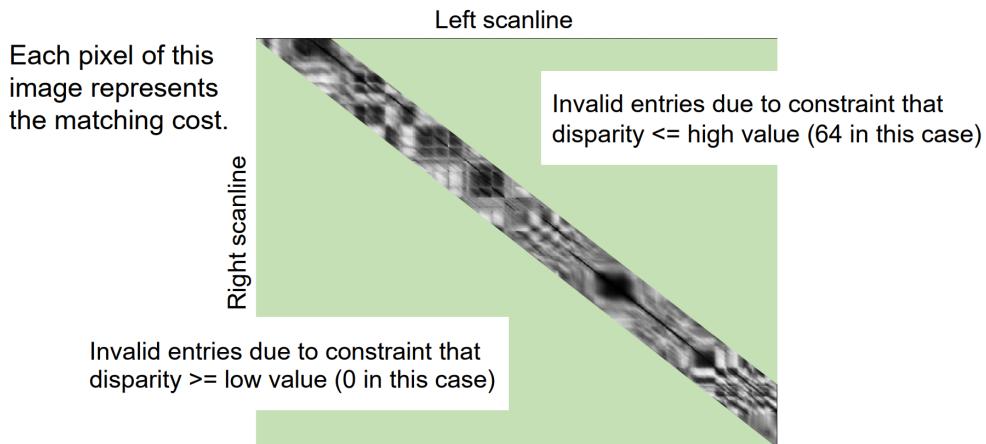
首先介绍局部匹配 (Local Stereo Matching)，在左图中取一个小窗口，然后在右图的同一行上移动同样大小的窗口，对每个位置计算匹配分数，比较两窗口的相似程度。特别地，由于右眼视觉比左眼偏右，我们可以忽略 $x > x'$ 的区域。做完一轮后，再交换两图再匹配一轮，以增强准确性。

对于匹配分数，最简单的模式是 **SSD (平方差的和)**，更复杂且精确的也有 ZNCC (排除二者亮度差异，再计算二者相关系数)

局部匹配会有一些失败的情形：半遮挡（仅有单目可见，从而无法匹配）；无纹理；特征重复；高光影响……因此，仅考虑局部最优无法很好地进行匹配，因此接下来介绍基于全局正则化进行的全局匹配。

不是单独看每个像素，而是让整个图像的匹配结果同时满足：Uniqueness (唯一性)、Ordering (顺序约束)、Smoothness (平滑约束)。有些地方会被违背，但是还是值得添加的。

我们扫描两侧的扫描线，得到任意两点之间的匹配程度，并排除 $x > x'$ 区域以及 $x + M < x'$ (这代表物体离双目过近) 的区域，得到如下的 $N * M$ 的条带：



接下来我们实际上就是需要找一条从左上到右下的“路径”，表示一个一一匹配关系。路径的移动代表了图像的匹配结果：在 (i, j) 节点，若其匹配成功，则移动至 $(i + 1, j + 1)$ 节点；若右侧遮挡，即左侧图像中像素在右侧中无匹配，则移动至 $(i + 1, j)$ (左图向下)，反之左侧遮挡则移动至 $(i, j + 1)$ 。综上，递推公式可以总结为：

$$C(i, j) = \min \begin{cases} C(i - 1, j - 1) + e(i, j), & \text{match ()} \\ C(i - 1, j) + \lambda_{\text{occ}}, & \text{occluded from right (\downarrow)} \\ C(i, j - 1) + \lambda_{\text{occ}}, & \text{occluded from left (\rightarrow)} \end{cases}$$

因此设计动态规划算法……

这种动态规划找最短路径的思路在其他领域也有广泛应用，例如在 Seam Carving 中，我们希望找到一条“能量最小”的缝 (seam)，删除它以无损缩小图像。其代价函数与 Stereo Matching 中的 DP 求最小路径问题形式相同。

同时兼顾考虑竖直方向的邻域，可以使用**整体能量函数**，使用 $E(d) = E_d(d) + \lambda E_s(d)$ ，其中 $E_d(d)$ 表示数据项，即所有像素的匹配代价之和，而 $E_s(d)$ 表示平滑项，用于惩罚相邻像素的视差。太慢？下采样。

也可以使用深度学习学习先验。

主动立体通过在场景上主动投射人工纹理来解决这个问题。

原理：使用一个投影仪向物体表面投射特定的光纹图案（如条纹、棋盘格、彩色编码），再使用相机观察该图案在物体表面的变形。

4.3 运动恢复结构 (SfM)

这是一个很难的任务，因为它 Input: point pairs from 2D to 2D; Output: 3D points and camera parameters

也即给出 x_{ij} ，找出满足 $x_{ij} \simeq P_i X_j$ 的相机参数和真实 3D 坐标。

更一般地，如果你对场景施加一个任意的线性变换 Q 并同时对相机矩阵施加它的逆变换 Q^{-1} 投影结果依然不会改变：

另一种方法是可以在相机参数上做一些假设，做一个 weak perspective 假设，直接将 z 坐标丢掉

从最简单的 Affair(仿射相机)开始，可以把正交投影矩阵写成： $P = \begin{pmatrix} K_{2D} & t_{2D} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_{3D} \\ 0 \end{pmatrix}$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & t_1 \\ a_{21} & a_{22} & a_{23} & t_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

可以在简单的相机上做消除一些歧义性。其中前两行仍旧满足正交性，于是消除了很多自由度。

可以写出 $AX + t$ 。如果将世界坐标系原点代入，可以得到投影坐标点 $t = (x, y)$ ，于是 t 就代表……投影。将 $\begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$ 代入得到，即为 X 轴投影。于是这个矩阵的每个参数都有其意义。

给定 m 张照片 n 个三维点，则

$$x_{ij} = A_i X_j$$

这是理想化假设（不一定都看得到），后面会修正。

我们会有 mn 个 2D 上坐标，表示第 j 个点在第 i 张图像上的投影。

在这个假设下事实上也是有歧义性的。可以乘一个仿射矩阵， Q 有 12 个自由度，也就是还有 12 个歧义性需要我们解决。

将对应点代入进去，会得到一系列的等式，其中有 $2mn$ 个已知量 (x_{ij}) ，有 $8m + 3n$ 个自由度 (A_i, t_i, X_j) 于是得到

$$2mn \geq 8m + 3n - 12$$

中心化。重新定义图像的原点。于是

$$\hat{x}_{ij} = A_i X_j + t_i + \frac{1}{n} \sum_{k=1}^n (A_i X_k + t_i) = A_i (X_j - \frac{1}{n} \sum_{k=1}^n X_k) = A_i \hat{X}_j$$

于是将 t 消掉了（将世界坐标系原点人为定到三维点中心上）

对于 $\hat{x} = A^T X$ 这个矩阵方程，其中观测矩阵 \hat{x} 为 $2m \times n$ 的矩阵，接下来只需要分解出 $A_{2m \times 3}, X_{3 \times n}$ 。注意到 \hat{x} 秩至多为 3，于是进行 SVD 分解： $\hat{x} = U \Sigma V$ ，像 Fundamental Matrix 一样作低秩分解，保留前 3 个特征值：

$$\hat{x} \approx U'_{2m \times 3} \Sigma'_{3 \times 3} V'_{3 \times n}$$

于是取 $A = U' \Sigma^{\frac{1}{2}} V'$, $X = \Sigma^{\frac{1}{2}} V'$ ，即为一组可行解。我们希望找到符合现实状况的可行解，消除仿射歧义性。我们需要再加一些约束，将放射矩阵 Q 消掉。

A 的前两行是正交的，即 $a_1 \cdot a_2 = 0, \|a_1\|^2 = \|a_2\|^2 = 1$ ，于是有 $3m$ 个约束，可以消掉 Q 的 9 个自由度。

$$(A_i Q)(A_i Q)^T = A_i (Q Q^T) A_i^T = I_{2 \times 2}$$

但这个约束不是线性的，可以用最小二乘将 $N = Q Q^T$ 求出来，进而 Cholesky 分解求解出 Q ，得到 AQ 和 $Q^T X$ ，即可。

接下来处理不可见问题。小子块中做分解 (19: 19) 诸葛分解逐个求解。找稠密子矩阵是 NP-完全的。

另外一个解决方法是增量方法。不断作三角化-配准的迭代，在实际过程中运用更广泛。

在一般性的相机中，变为 $x_{ij} \cong P_i X_j$ ，如果没有任何配准信息，可以容忍一个 15 自由度（缩放一致）的 Q ，于是可解当

$$2mn \geq 11m + 3n - 15$$

两相机情况下，需要七个点。pipeline:

……估本征矩阵作为启动，不断迭代即可实现左脚踩右脚。但这会导致误差累积，因此需要再作 Bundle adjustment，进行优化，即使到现在，也需要很长时间

Photo Tourism。提 feature (高维空间中 KD-tree 找最近邻)，估 Fundamental Matrix。用 EXIF 作为初始内参矩阵，解得两张照片的 R, t ，triangulation 初始化点

匹配现有模型，然后作 Bundle adjustment

回环检测作为……

好难……

4.4 MVS

通过 calibration 好的相机，重建完整几何。SfM 只能求出稀疏点，而希望得到稠密点的表达，需要在 SfM 之后接一个 MVS。

第五章 几何

5.1 几何表示

几何这一模块从最基础的问题开始——如何在计算机上表示几何？表示方法可以分为两类，**显式的点云、多边形网格（及其衍生）等和隐式的水平集（根据与表面的距离定义函数）、代数曲面（直接用多项式等表示）等。**每种表示都有着广泛的应用，本节重点介绍**多边形网格**（polygon mesh）。

从最简单的三角形网格讲起。暂且不讨论如何将点划分为三角形（这一部分将会在后续内容中介绍），我们先关注如何**存储**三角形网格。最直接的方式是记录所有顶点的位置，再通过列表保存每个三角形或边所包含的顶点编号。然而，这种表示方式虽然简单，却无法方便地获取三角形之间的**邻接关系**，因此在执行遍历、光滑、细分等操作时会遇到困难。

为了解决这一问题，我们引入了一种更高效的拓扑表示方式——**半边数据结构**。该结构将每条边拆分为两条**有向半边**，每条半边分别记录其起点、对应的反向半边、在同一面中的下一条半边、所属的“无向边”，以及所属的面等信息。通过这些相互引用的指针关系，可以轻松地在顶点、边和面之间进行遍历，从而有效支持后续的几何处理与拓扑操作。

四边形网格实现原理类似，且因其易于存储与编辑、易于参数化等特点，被更广泛地应用于人体、面部建模等实际任务中。

多边形网格尽管结构简单且易于存储，但表现曲面需要大量面片，使得效率较低，而接下来介绍的**细分曲面**提供了一种“用少量控制点生成无限平滑曲面”的方式：从一个粗糙的控制网格出发，逐层插入新顶点并调整其位置，最终收敛为连续光滑曲面。下面介绍两种细分方式：

第一种 **Catmull–Clark 细分**，从粗糙的四边形网格出发，向每个四边形中添加 1 个**面点**（四个顶点的平均）和 4 个**边点**（对于每条边，取两个顶点和邻接面的面点的平均），再根据以下公式调整原四边形顶点的位置：

$$v' = \frac{1}{n} \text{AVG(face points)} + \frac{2}{n} \text{AVG(edge midpoints)} + \frac{n-3}{n} p \quad (n \text{ 表示顶点连接的边的个数})$$

如此从几何直观上，新顶点位置会向邻近面的中心略微靠拢，在效果上更为平滑。之后将**新顶点、面点、边点**顺次连接，可以将分辨率扩大 4 倍（相当于从 2×2 变为了 3×3 ），即实现了曲面的细分。

第二种是 **Loop 细分**，是针对三角形网格的算法。其思想与 Catmull–Clark 细分相似，对于三角形网格的每条边添加**边点** $v = \frac{3}{8}(v_0 + v_1) + \frac{1}{8}(v_2 + v_3)$ ，其中 v_0, v_1 为其对应顶点， v_2, v_3 为其对角

点。特别地，若其为边界边，则直接 $v = \frac{1}{2}(v_0 + v_1)$ 即可。随后更新顶点：

$$v' = (1 - n \cdot u) v + u \sum_{i=1}^n v_i, \quad \text{平滑系数 } u = \begin{cases} \frac{3}{16}, & n = 3 \\ \frac{3}{8n}, & \text{otherwise} \end{cases}$$

最后进行顺次连接即可。

最后讲**网格的参数化**。我们希望在不改变拓扑关系的前提下，将三维流形表面映射到二维平面上，从而实现例如地图绘制、纹理展开等应用。这种“展平”过程类似于世界地图的各种投影方式：例如，墨卡托投影保角度，朗伯投影保面积。同样地，在网格参数化中，我们也可以选择保留三维表面的某些几何性质——如保长度、保角或保面积——但显然，展平过程仍然不可避免地会**损失部分几何信息**。根据参数域的不同，常见的参数化方式还包括平面参数化、球面参数化以及基域参数化（即在简化几何形状上展开）等。

这里介绍一种典型的平面参数化方法——**弹簧系统模型** (Spring System)。我们假设每条网格边 (i, j) 就像一根弹簧，希望它在二维展开后仍保持与原始长度接近。于是定义能量函数：

$$E = \frac{1}{2} \sum_{i=1}^n \sum_{j \in N_i} \frac{1}{2} D_{ij} \|t_i - t_j\|^2,$$

其中 D_{ij} 为弹簧系数， $t_i = (u_i, v_i)$ 为平面上的顶点坐标。通过对能量函数求导并令其为零，可以得到每个顶点的平衡条件：

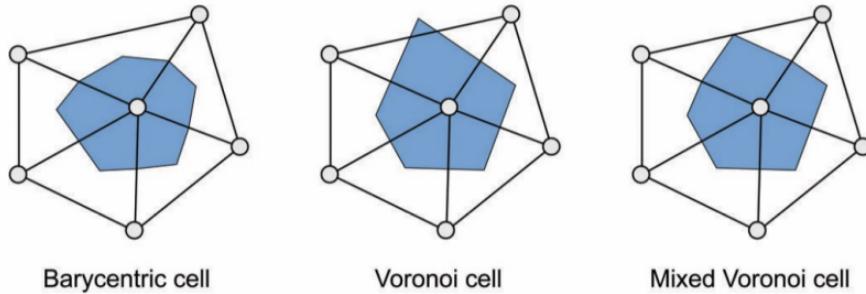
$$\frac{\partial E}{\partial t_i} = \sum_{j \in N_i} D_{ij} (t_i - t_j) = 0 \Rightarrow t_i = \sum_{j \in N_i} \lambda_{ij} t_j, \quad \lambda_{ij} = \frac{D_{ij}}{\sum_{k \in N_i} D_{ik}}.$$

将所有顶点的方程整理为矩阵形式，可得 $At = 0$ ，其中矩阵 A 由 λ_{ij} 组成。但需注意，该线性系统存在平凡解 ($t_i = 0$)，因此在求解时必须加入额外约束条件，例如固定边界点等（如此可以写成 $At = b$ ）。最终即可解出各顶点的平面坐标 $t_i = (u_i, v_i)$ ，实现平滑的网格参数化。

5.2 几何处理

几何处理是针对构建好的网格的一些几何操作。在具体介绍处理类型之前，需要先简单介绍**离散微分几何**，其提供了一种语言，用来描述形状的局部性质（例如曲率、法向、面积、梯度等）。在连续情况下，这些性质通过微分方程或偏导数定义；而在离散情况下，我们用网格顶点、边、面上的几何量来近似它们。

主要介绍其中的几个重要概念：**局部平均区域**，其定义了离散的“积分邻域” $\Omega(x)$ ，常见的定义方式有三种：**重心单元** (Barycentric cell)，即顶点区域由相邻三角形重心连线围成，其简单易实现，但精度较低；**Voronoi 单元** (Voronoi cell)，即顶点区域由邻边中垂线组合而成，其有严格的几何意义，十分精确，但对钝角三角形会出现负面积等问题；**混合单元** (Mixed Voronoi cell) 结合了二者的优点，对 Voronoi 单元的钝角三角形改用重心划分，是目前几何处理中常用的折中方案。三种划分的效果见下图：



在顶点处的**单位法向量**也由周围区域的平均加权给出：

$$\mathbf{n}(v) = \frac{\sum_{T \in \Omega(v)} \alpha_T \mathbf{n}(T)}{\left\| \sum_{T \in \Omega(v)} \alpha_T \mathbf{n}(T) \right\|}$$

其中 α_T 代表权重系数，最为标准的实现是 $\alpha_T = \theta(T)$ ，即根据夹角加权。

设在每个顶点 x_i 上定义了一个函数值 f_i ，我们也可以在离散网格上定义**梯度**运算（以三角形网格为例）。在每个三角形内，由于在每个三角形内 f 通常被近似为分片线性函数：

$$f(x) = \alpha f_i + \beta f_j + \gamma f_k \Rightarrow \nabla_x f(x) = f_i \nabla_x \alpha + f_j \nabla_x \beta + f_k \nabla_x \gamma$$

根据几何关系可得：

$$\nabla_x \alpha = \frac{(x_k - x_j)^\perp}{2A_T}, \quad \nabla_x \beta = \frac{(x_i - x_k)^\perp}{2A_T}, \quad \nabla_x \gamma = \frac{(x_j - x_i)^\perp}{2A_T}$$

其中 $(\cdot)^\perp$ 表示垂直向量， A_T 为该三角形的面积。因此，离散梯度可表示为：

$$\nabla_x f(x) = \frac{1}{2A_T} [f_i(x_k - x_j)^\perp + f_j(x_i - x_k)^\perp + f_k(x_j - x_i)^\perp]$$

最后是**拉普拉斯算子**。回忆二维图像的拉普拉斯算子，是周围像素点的加权平均，可以将其自然推广到网格中：

$$(\Delta f)_i = \sum_{j \in \Omega(i)} \omega_{ij} (f_j - f_i)$$

其中 ω_{ij} 表示邻接顶点的权重，比较朴素的方式是 $\omega_{ij} = \frac{1}{N_i}$ ，即标准拉普拉斯算子，但由于网格不规则，这样的效果往往不好，经过（复杂的）数学推导，实际上效果最好的参数是余切权：

$$\omega_{ij} = \cot \alpha_{ij} + \cot \beta_{ij}, \quad \alpha_{ij}, \beta_{ij} \text{ 为这条边所对两个三角形的对角}$$

直接运用即可。有了上述的数学基础，就可以完成一些简单的几何处理任务，下面介绍几例。

网格平滑与图像平滑类似，依赖拉普拉斯算子多次迭代实现平滑操作：

$$\frac{\partial f(x, t)}{\partial t} = \lambda \Delta f(x, t) \Rightarrow (\text{迭代更新}) x_i \leftarrow x_i + h \lambda \Delta x_i$$

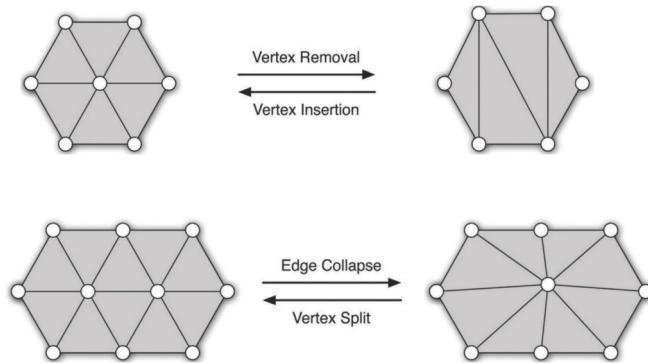
传统的**网格编辑**中，可以通过修改顶点位置来实现整体变形，但这样会丢失局部细节（表面的皱褶/起伏等）。因此引入**细节保持网格编辑**，设定空间约束点，并每次重新更新表面，使

其在满足约束的同时尽可能保持局部细节。具体而言，与泊松图像编辑思路类似，我们最小化如下公式：

$$\tilde{V}' = \arg \min_{V'} \left(\|L(V') - \Delta\|^2 + \sum_{i \in C} \|v'_i - u_i\|^2 \right)$$

其中 $\|L(V') - \Delta\|^2$ 促进保持原始细节，而 $\sum_{i \in C} \|v'_i - u_i\|^2$ 促进满足编辑约束。

网格简化的目标是把一个复杂的网格简化成一个面数更少但形状尽量接近原模型的版本，简化方法主要分为**顶点删除**和**边坍缩**（保持整体拓扑结构前提下将一条边对应的两个顶点合并，在半边结构中同样有半边坍缩），如下图：



在进行网格简化时，我们需要决定**每次坍缩哪一条边**。研究者提出了一种高效方案：在每次操作前，计算**每条边折叠后的几何误差代价**，并优先选择代价最小的一条进行坍缩。这一算法被称为**QEM 算法**。

其核心思想是：每次折叠生成的新顶点应尽可能贴近原始网格的局部几何形状。具体而言，新顶点的位置应**最小化其到原邻域所有三角面平面的 L_2 距离之和**。定义误差函数如下：

$$E(\mathbf{v}) = \sum_i h_i^2 = \sum_i \mathbf{v}^T (\mathbf{n}_i \mathbf{n}_i^T) \mathbf{v} = \mathbf{v}^T \mathbf{Q} \mathbf{v},$$

其中， n_i 为第 i 个相邻三角面的单位法向量， Q 称为**二次误差矩阵 (QEM)**。算法流程如下：预先为每个顶点计算其 Q 矩阵，然后对所有候选边 (v_1, v_2) 计算合并后的误差 $Q_{new} = Q_{v_1} + Q_{v_2}$ 及其最小代价点。每次选择误差最小的边进行坍缩，并更新受影响区域的代价，迭代执行直至达到预期的简化程度。

5.3 几何重建

本节介绍在已获得多个独立的 3D 点云前提下，如何完成完整世界的重建。

首先，第一步需要进行点云的**配准 (Registration)**，即找到一个合适的变换，将两个不同的点云进行对齐。在此介绍**ICP (Iterative Closest Point)** 算法，其可以适用于任意维度，下面以二维为例。

从名字可以看出，这是一个迭代算法，因此首先需要找到合适的初值。此处利用**主成分分析(PCA)**，找出两组点云大致的变化方向，由此得到初步的旋转矩阵。具体步骤为：**去中心化**（将每个点减去质心坐标，得到 $k \times n$ 矩阵 P ，相当于将重心移至原点） \rightarrow **计算协方差矩阵**

$$M = \frac{1}{n} P P^T = \begin{pmatrix} \text{cov}(X, X) & \text{cov}(X, Y) \\ \text{cov}(Y, X) & \text{cov}(Y, Y) \end{pmatrix}$$

其中，协方差表示两个坐标间的相关性，理论上主方向对应的方差 ($\text{cov}(T, T)$) 最大。随后进行**特征值分解** $M = V \Lambda V^T$ （或直接对 P^T 进行 SVD 分解），其中**最大特征值对应的特征向量**即代表数据变化最显著的方向，即主方向。得到源点云与目标点云的主方向 V_S, V_T 后，即可求出初始旋转矩阵 $R = V_T V_S^T$ （表示将 S 旋转至 T ）。平移向量亦可由下方推导得出：

$$p'_i = c_T + R(p_i^* - c_T) = c_T + R(p_i - c_S)$$

其中 c_S, c_T 分别为源点云与目标点云的质心， p_i^* 为质心对齐后的点。于是

$$[R, t] = [R, c_T - R c_S]$$

即得到迭代初值。

接下来，对于点云 P 中的每个点，在 Q 中寻找其**最近邻点**（在此过程中需剔除异常匹配点，一般取中位数乘以常数 k 为阈值），然后通过**SVD** 优化能量函数：

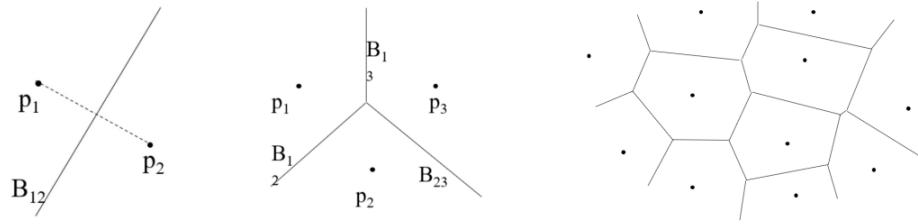
$$E = \sum_i \|Rp_i + t - q_i\|^2$$

这一过程与 PCA 在形式上十分相似（不同点在于此处已有对应点信息，因此可计算**互协方差矩阵**）。去中心化后，将配对点形成的矩阵相乘得到 $M = P Q^T$ ，对 M 作奇异值分解，得 $M = U \Sigma V^T$ 。经过数学推导可得，要求的旋转矩阵即为 $R = V U^T$ ，进而即可更新 R 与 t 。如此迭代直至收敛，便完成了 ICP 点云配准的过程。

接下来我们希望在点之间建立拓扑关系——也就是从点云**重建表面** (Mesh)，进而进行光照、纹理、阴影等计算。重建表面过程可以手工完成，但对复杂的点云结构，设计合适的算法可以加快效率。这样的算法主要分为两种，**三角剖分**和**隐式重建**，下面分别介绍。

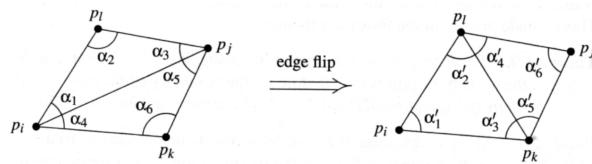
在进行三角剖分之前，我们需要解决一个更基础的问题：**点是如何划分空间的？**事实上，不同的点在空间中拥有各自的“影响区域”，这一划分方式可以通过**Voronoi 图**来描述，其是由相邻两点间垂直平分线组成的划分，或者更直观地，每个顶点以相同的速率扩张，直至二者相遇则停止，扩张成的区域就是这个点的影响区域。Voronoi 图示意如下图：

$$V(p) = \{x \mid d(p_i - x) \leq d(p_j - x), \forall j \neq i\}$$



然而, Voronoi 图在实际计算中较为复杂, 因此通常考虑其几何对偶形式——Delaunay 三角剖分。二者的对偶关系体现在: 若 Voronoi 图中两个区域 P_i 和 P_j 共享一条边, 则在 Delaunay 三角剖分中就存在一条连接 P_i 与 P_j 的边。Delaunay 三角剖分具有良好的“空圆性质”, 即对于剖分中的任一三角形, 其外接圆内不包含其他采样点。该性质保证了三角形的内角尽量均衡, 避免出现过于瘦长的三角形。换句话说, Delaunay 剖分倾向于**最大化最小角**, 从而提升网格的整体质量与稳定性。

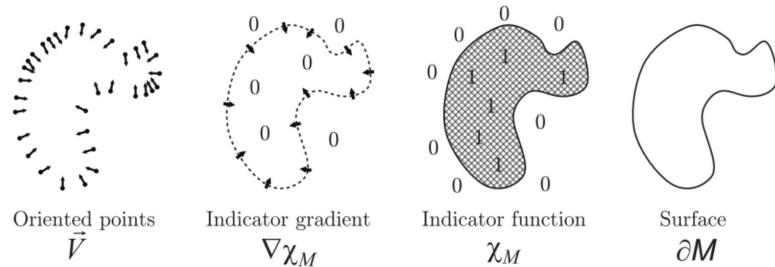
在具体计算层面 (Delaunay 三角剖分算法) 中, 我们只需考虑一种基本操作: **边的翻转**, 如下图所示:



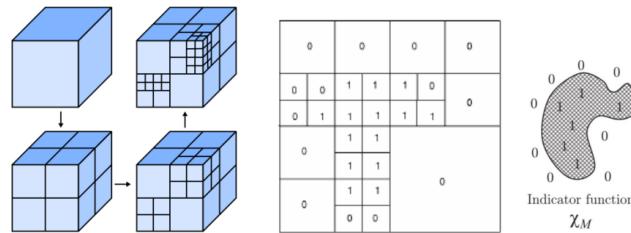
若翻转后能够使局部的角度分布更加合理 (即最小角度变大), 则执行该翻转操作, 从而保证局部最优。具体而言, 可以先随机生成一个初始的三角剖分, 然后不断寻找一条在翻转后能改进角度分布的边并进行翻转, 直至整个剖分达到全局最优。更高效的做法是在插入新点的同时, 对新产生的内部边进行检测 (实际上最多需要检测三条边), 并对其中可以优化的边进行翻转, 从而实现对 Delaunay 三角剖分的**动态维护**。这样的分割方式和算法可以自然地推广到三维情况。

Delaunay 三角剖分有很广泛的应用, 在此举其中一例: 假设我们有一组二维平面上的采样点, 如何估算那些**不在采样点集合**中的点的高度 (也即, 从离散点重建连续表面)? 只需做三角剖分, 再将每个采样点按其高度“抬高”到三维空间中, 在平面网格基础上, 构建出一个三维的“山地网格”。随后, 在三角形内部对未知点线性插值即可。

但在点较为稠密时, 显式的三角剖分效率可能过低, 此时就需要用到**隐式重建**, 其中比较重要的是**泊松表面重建**: 给定带法向量 (这可以通过**主成分分析找最短轴**实现) 的采样点, 希望构建一个**隐式函数**, 内部值为 1, 外部值为 0, 进而构建表面, 如下图:

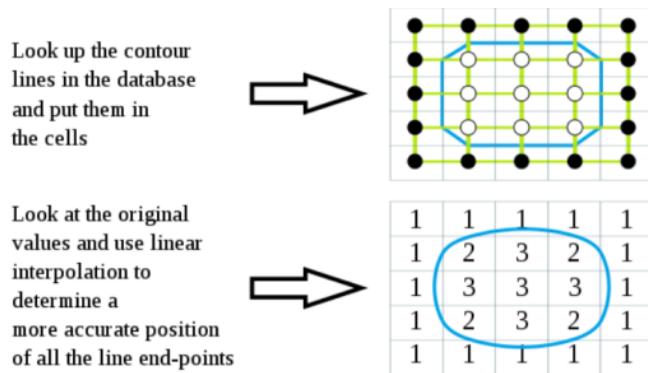


具体而言，我们获得法向后，需要对空间内/外部进行标记。朴素的方法是全体素标记，而一个更节省内存的方法是利用**八叉树**结构，仅在点云表面附近的区域进行细分，在距离较远的区域保持粗糙，如下图：



随后利用与泊松编辑类似而又略有不同的方式离散地求解 Poisson 方程 $\Delta \chi = \nabla \cdot V$ ，最终对每个点得到一个内外部程度 χ 。最终，我们需要从体素网格提取三角网格，此处运用的主要是 **Marching Cubes 算法**。

首先，我们对 $\chi(x, y, z)$ 进行二值化，从而筛选出表面需要穿过的小立方体（也就是顶点既有 0 也有 1 的）。根据对称性，这样的拓扑不同的小立方体仅有 15 种，于是只需构建一个哈希表，输入为一个 8 位二进制数（表示哪些是 0，哪些是 1），返回一个 **12 位二进制数** 表示哪些边需要被插值，以及一个数组表示边的连接顺序即可。如下图，还可以将原本的连接中点改为线性插值以增强平滑性。



最后，我们还需要计算表面法向量（这是为了后续生成正确的光照信息）。计算方法是采取中心差分计算 $G_x = \frac{D(i+1,j,k) - D(i-1,j,k)}{\Delta x}$, G_y, G_z ，再组合成 $G(i, j, k)$ 并进行归一化。

总而言之，通过取立方体-二值化-查表-线性插值-计算表面法向量，就可以组合成最终的表面网格。有很多种方法可以优化这一过程，例如记录前一层的梯度、插值等信息进行复用提高效率等。

最后一步，我们希望做**模型匹配**（用较小参数量表示数据，例如三维平面 $Ax + By + Cz = D$ ）这其中的关键挑战是数据中存在的**离群点**（outlier），于是并非所有点都符合模型参数，因此不能简单地用最小二乘法直接拟合，于是我们可以用此前讲过的 **RANSAC** 取出噪点，再将筛选出的 inliers 进行最小二乘拟合。

5.4 几何变换

首先介绍基础的几何变换。在二维坐标中，平移、旋转、缩放与斜切都可以用矩阵表述；为使平移也能写成矩阵乘法，需要引入**齐次坐标**，并区分点 $(x, y, 1)^T$ 与向量 $(x, y, 0)^T$ 。这些概念可以自然推广到三维，其中三维旋转较为复杂。单独绕 $x/y/z$ 轴的旋转较简单，但一般的绕任意轴旋转可视为：将向量分解为沿轴方向的分量与其法平面内的分量，并在法平面内完成二维旋转。由此可得绕单位轴向量 \mathbf{a} 旋转 θ 的矩阵：

$$\mathbf{R} = \mathbf{a}\mathbf{a}^T + \cos\theta(\mathbf{I} - \mathbf{a}\mathbf{a}^T) + \sin\theta \mathbf{a}^*,$$

其中 \mathbf{a}^* 为 \mathbf{a} 的对偶矩阵，满足 $\mathbf{a}^*x = \mathbf{a} \times x$ ，该式即 **Rodrigues 旋转公式**。

在实际计算中，亦可采用更直接的方法——基于**欧拉角**的旋转：将绕任意轴的旋转分解为绕三个坐标轴的基本旋转，

$$R = R_z(\gamma) R_y(\beta) R_x(\alpha).$$

这种表示虽然直观，但存在局限：其插值不具线性性，且基本旋转矩阵之间不满足交换律。

上述变换在动画中具有基础性作用，其核心思想是：通过一系列相互连接的铰接刚体执行连续且相关的局部变换，从而在整体上产生复杂的宏观动作。其具体机制将在后续的动画部分中进一步展开。

在现代计算机图形学中，常使用诸如 OpenGL 等图形库来完成几何变换，使一个三维点经历从物体空间、世界坐标空间、相机空间、裁剪空间到屏幕空间转换的完整流水线。

首先，通过 Model Transform（模型变换）将点从局部模型坐标变换到 World Space。对应的 `model2world` 矩阵由平移、旋转与缩放组成，其作用是将模型正确地放置于整个世界坐标系中。

接着应用 View Transform（视图变换），将世界坐标转为 Camera/Eye Space。OpenGL 默认摄像机位于坐标系原点并朝向 $-Z$ 方向，因此我们需根据 LookFrom-LookAt 以及 Up 向量构建摄像机坐标系，并将其对齐到标准的 (x, y, z) 轴方向。

随后执行 Projection Transform（投影变换），将点映射到 Clip Space，可选择正交投影或透视投影。正交投影可简单理解为保留 x, y 并丢弃 z （实际仍可结合 z-buffer 使用）；而透视投影的核心是将坐标按

$$x' = \frac{nx}{z}, \quad y' = \frac{ny}{z}$$

缩放到 near 平面。然而这样会使深度关系丢失，因此必须构造一个适当的 z' （深度值），使得 near 与 far 平面的深度最终都能映射到 $[-1, 1]$ 的标准区间。这一步的推导较为繁琐，我暂时还没搞懂，等我搞懂了再补。

第六章 图像生成模型

图像生成任务在数学上即是训练一个网络 G , 将一个高斯分布 z 映射到目标分布 $G(z)$. 我们希望找到一个好的生成模型, 其同时具备生成质量高、采样速度快、训练难度低等特点, 但很遗憾, 尽管图像生成模型在近十年飞速发展, 我们还没有找到同时满足上述三者的模型。具体而言, **变分自编码器**生成质量低, **生成对抗网络**训练难度高, 而新兴的**自回归模型**和**扩散模型**训练速度较慢(但由于算力的飞速提升, 慢并不会成为发展的主要阻碍), 其特征归纳如下图:

	VAE	CAN	自回归模型	Diffusion
直接计算 $p(x)$	✗	✗	✓	✗
使用隐变量 z	✓	✓	✗	✓
生成速度快	✓	✓	✗	✗
训练稳定性	✓	✗	✓	✓
高质量图像	✗	✓	✓	✓

接下来就详细介绍一下这四种模型。

6.1 变分自编码器

自编码器 (自回归编码器) 顾名思义, 就是训练**编码器** E 提取图像 x 的特征 z , 由于我们无法得到真实特征 z' 作监督, 于是使用无监督学习, 用编码出的 z 通过**解码器**重建出 \hat{x} , 进而优化**重建损失** $\|x - \hat{x}\|^2$ 即可。这样的模型可以很好地完成重建任务, 但在生成任务中, 训练后的模型依然需要输入 x 才能得到 \hat{x} , 从而无法完成, 其本质是 z 的生成空间是混乱而不连续的, 所以我们的思路是对 z 的生成空间进行限制, 构建连续(进而可以插值)的潜空间, 这便是**变分自编码器(VAE)**的核心思想。

带着这个思想, 我们尝试限制 z 为一个高斯分布 $z \sim N(0, I)$, 并尝试训练一个合适的解码器 θ , 使得其可以在训练后输出一张“合理”的图像, 也即对于真实数据 x , 最大化 $p_\theta(x)$. 但这个优化

过程事实上是困难的，因为在下面的全概率公式中：

$$p_\theta(x) = \int p_\theta(x|z)p(z)dz$$

虽然对于特定的解码器 θ , $p_\theta(x|z)$ 是已知的，但我们无法计算所有对 z 的积分，换句话说，在下面的贝叶斯公式中

$$p_\theta(x) = \frac{p_\theta(x|z)p(z)}{p_\theta(z|x)}$$

$p_\theta(z|x)$ 这一项，也即给定 x ，哪个高斯分布的 z 对应它是未知的。VAE 的解决方案是，不会就学——尝试用编码器 ϕ 学习该未知分布 $q_\phi(z|x) \approx p_\theta(z|x)$ ，具体而言，我们的训练目标如下：

$$\begin{aligned} \log p_\theta(x) &= \log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)} + \log \frac{q_\phi(z|x)}{p_\theta(z|x)} \text{(这很有技巧性，但也很自然)} \\ &= E_{z \sim q_\phi(z|x)}[\log \frac{p_\theta(x|z)p(z)}{q_\phi(z|x)}] + E_{z \sim q_\phi(z|x)}[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}] \\ &\quad (\text{这一步利用了 } E_{z \sim q_\phi(z|x)}[\log p_\theta(x)] = \int q_\phi(z|x) \log p_\theta(x) dz = \log p_\theta(x)) \\ &= E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] + E_{z \sim q_\phi(z|x)}[\log \frac{p(z)}{q_\phi(z|x)}] + E_{z \sim q_\phi(z|x)}[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}] \\ &= E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z)) + D_{KL}(q_\phi(z|x), p_\theta(z|x)) \text{(KL 散度定义)} \\ &\geq E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z)) \end{aligned}$$

(KL 散度非负，因此可以将第二个无法学到的 KL 散度项消去，得到一个可学习的下界)

最终，我们（很不容易地）得到了我们需要最大化的 ELBO 下界：

$$E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z))$$

接下来让我们实际代入计算一下：

$$\begin{aligned} \log p_\theta(x) &\geq E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z)) \\ &= -\frac{1}{2\sigma^2} E_{z \sim q_\phi(z|x)} \|x - \hat{x}\|^2 - \frac{d}{2} \log(2\pi\sigma^2) - \frac{1}{2}(1 + \log \sigma_{z|x}^2 - \sigma_{z|x}^2 + \mu_{z|x}^2) \\ &\Rightarrow L \propto \|x - \hat{x}\|^2 + \beta(1 + \log \sigma_{z|x}^2 - \sigma_{z|x}^2 + \mu_{z|x}^2) \end{aligned}$$

我们最终得到了损失函数的形式，这个形式远比推导更好理解——前一项代表**重建损失**，也就是“生成的图像像不像训练数据”，后一项代表**KL 损失**，即训练出的特征 z 是否与先验设定的高斯分布类似（几何上来看，代表后验分布 $q_\phi(z|x)$ 的高斯分布椭圆与单位圆是否接近）。

但在训练中还存在一个问题，我们需要随机从高斯分布中采样一个 z 已得到后续的 \hat{x} ，而这一过程会打断梯度下降更新，解决方案是**重参数化技巧**，即改变随机变量的参数化方式： $z = \mu_\phi(x) + \sigma_\phi(x) \cdot \epsilon$, $\epsilon \sim N(0, I)$ ，使其可导。

训练完成后，采样 $z \sim p(z)$ 再采样 x 即可（实际应用中，可以直接设定 $x = \mu_{x|z}$ ）。以上模型还支持图像编辑： z 的各个维度是相互独立的，于是将输入图像通过编码器得到特征 z ，修改一些特定维度得到 z' ，再通过解码器输出图像即可。

但是我们也提到了，VAE 的效果其实不尽如人意，其主要原因有两点：重建损失实则代表让模型生成平均意义上最可能的像素值，于是当训练数据存在多模态分布（例如一只猫的不同姿态），取平均结果就会模糊；第二点是高斯分布的假设过于理想，在真实复杂的潜空间中往往丢失结构信息，导致生成样本单一、缺乏细节（当然，还有一些小问题，比如需要调整 β 平衡两个损失的权重；以及优化下界与优化整体的差距等）。

这两个问题中，前者是 VAE 甚至 AE 本身的结构性问题，很难解决（只能融合其他算法，比如后面介绍的 cVAE-GAN），而后者可以被多种方法优化，其中比较经典的一种是 **VQ-VAE**。它的主要优化思路是，AE/VAE 编码得到的向量是连续向量，也就是向量的每一维都是浮点数。而改进后的 VQ-VAE 将图片编码成离散向量，这样会使图像更加自然（解释：对于标签，如职业/性别/年龄，我们的目标一般是离散的，而不会说这个人性别是 0.5，年龄是 0.6……同时，其限制了解空间，更容易学习）

但仍有一个问题，训练的网络默认输入满足 **连续分布**，也即若标签是非连续的（例如职业等），会带来语义的扭曲。为此，VQ-VAE 借鉴了 NLP 中的**词嵌入**思想——即为每个离散标签构建一个可学习的“嵌入向量”，通过查表操作，将离散索引映射为连续空间中的向量表示。具体而言，编码器首先输出连续潜向量 $z_e(x)$ ，随后通过向量量化（**Vector Quantization**）选择与其最接近的嵌入向量 e_k 作为离散表示 $z_q(x)$ ，再由解码器将其还原为重建图像 \hat{x} 。其中，由于量化一步不可导，在梯度计算过程中直接将取最近邻近似为**恒等映射**，将梯度原样反传，整体过程既保留了离散潜空间的可解释性，又恢复了连续空间的可微性，使模型能够端到端训练。

这样的设计已经能很好地在现实中实现重建任务了，而对于生成任务，由于生成预测内容从空间较大的 RGB 像素值变为了尺寸较小的字典索引，较简单的 VAE 有较大的改善，但效果仍没有达到我们的期望，究其原因是因为我们用的解码器能力不足，我们会在后面解决这个问题。

6.2 生成对抗网络 (GAN)

生成对抗网络的思想是：希望生成器的生成的 $G(z)$ 所在的生成分布 p_G 尽可能接近真实分布 p_{data} ，而所谓的“接近”即是尽量让二者之间难以辨别。于是我们引入**判别器 D**，区分图像的真假；同时训练生成器 G ，目标是让 D 无法分辨图像的真假，二者具有相反的训练目标，即实现**对抗网络**。

形式化来说，判别器 D 的目标是使真实数据 $D(x)$ 的预测接近 1，并使生成数据 $D(G(z))$ 的预测接近 0；而生成器 G 的目标是使 $D(G(z))$ 接近 1。写进同一个式子，即要求

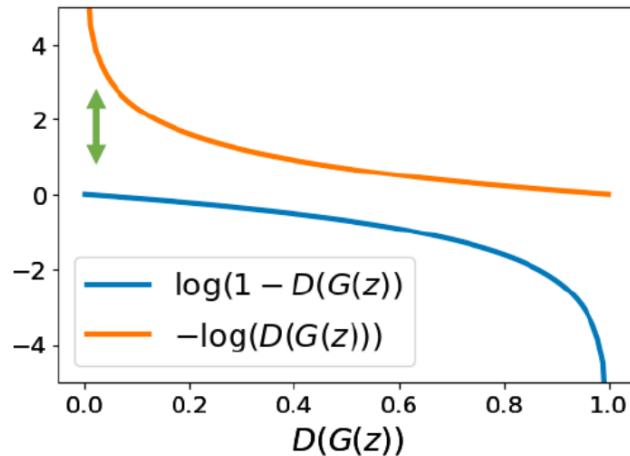
$$\min_G \max_D (E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} [\log(1 - D(G(z)))])$$

于是朴素的训练方案便是交替固定其中一个，训练另一个即可。可以证明，全局的纳什均衡是当 $p_G = p_{data}$ 时， $\forall x, D(x) = 0.5$ ，也即获得一个无比强大的生成器，使得判别器完全无法区分。但很遗憾，这只是美好的想象，实际训练中存在很多问题，而最显著的便是判别器收敛速度过快，导

致 $D(G(z))$ 长期趋近于 0，生成器无法有效学习。对此的解决方案有两种：采用**非饱和的 GAN 损失函数**，即修改为优化如下的函数：

$$\min_G \max_D (E_{x \sim p_{data}}[\log D(x)] + E_{z \sim p(z)}[-\log D(G(z))])$$

与之前的函数不同，当 $D(G(z))$ 接近 0 的时候，梯度很大，从而在一开始（生成图像质量很差时）从而鞭策 G 在开始时迅速更新。两种函数的对比如下图所示：



第二种被称为 **LSGAN**，是使用了 L2 损失函数代替了 GAN 的损失函数。其具体形式为

$$L_D = E_x[(D(x) - 1)^2] + E_z[D(G(z))^2], \quad L_G = E_z[(D(G(z)) - 1)^2]$$

其同样可以鞭策 G 在开始时迅速更新，同时，其可以直接输出实值函数而不需要经过 sigmoid 层分类，可以防止 sigmoid 层使梯度值显著减小，使训练更快、更稳定。

但这样的设计仍然是经验性的，而 **Wasserstein GAN** 则用数学推导给出了一套理论上正确的损失函数：

$$\min_G \max_{D \in 1-\text{Lipschitz}} E_{x \sim p_r}(D(x)) - E_{z \sim p_z}(D(G(z)))$$

这其中最重要的就是加入了对判别器 D 的 Lipschitz 约束，对 D 的参数范围进行限制，防止 D 输出实值函数后某一项陷入无限大/无限小而使训练受阻。具体而言，需要约束 $|D(x_1) - D(x_2)| \leq \|x_1 - x_2\|$ ，或者说转化为一个充分条件： $\|\nabla_x D(x)\| \leq 1$. 实际在 GAN 中，这一点可以通过两种实现方式：

第一种，约束对 x 的梯度在 GAN 中即体现为**约束权重**，因此直接暴力地将 $wclip$ 到 $(-0.01, 0.01)$ 之间以保证梯度范围有限；第二种，在损失函数中增加一个惩罚项： $L_D = -E_x[D(x)] + E_z[D(G(z))] + \lambda(\|\nabla_x D(x)\| - 1)^2$ ，后一种更为平滑稳定，被称为 **WGAN-GP**。

以上是在目标函数层面的改进，而 **DCGAN**（深度卷积生成对抗网络）则在网络结构上进行了优化，用**卷积层/反卷积层**替代全连接层。具体而言，生成器 G （向量 \rightarrow 图像）通过反卷积实现上采样，判别器 D （图像 \rightarrow 向量）通过卷积实现下采样，从而得到可用于有效特征提取和插值的向量表示。

ProGAN 在此基础上进一步优化了训练策略：从 4×4 的小模型开始训练，收敛后逐步增加层数，实现渐进式扩展，以提升高分辨率训练的稳定性。

进一步地，可将随机噪声映射为 **风格变量**，并在每层通过 AdaIN 注入不同风格（高层控制全局结构，低层控制局部细节），实现对生成图像的可控性——这正是 **StyleGAN** 的核心思想。

为 GAN 加入条件 y ，使生成器学习 $G(z, y)$ ，判别器学习 $D(x, y)$ ，从而生成符合指定目标的图像，这被称为 **cGAN**（**条件生成对抗网络**）。若进一步让判别器同时承担分类任务，与生成器协同优化生成与分类两项目标，则得到 **ACGAN**；需要注意在分类支路上，生成器与判别器是协作而非对抗关系。cVAE-GAN（待补）；最后，在工程实现上，通过显著提升 batch 大小与网络通道数，并将条件 y 注入到生成器的每一层（既直接作为层输入，也在 BN 层重参数化时使用 $\gamma(y), \beta(y)$ 进行调制），同时在判别器输出端引入特征向量与 y 的点积项以增强语义匹配能力，从而诞生了在实际应用中表现卓越的 **BigGAN**。

上述优化大多都在解决 GAN 训练不稳定的问题，使得其生成效果好的优点得以真正在任务中发挥。从 2014 年开始的 6-7 年，是 GAN 的爆发时期，直到更先进的模型出现……

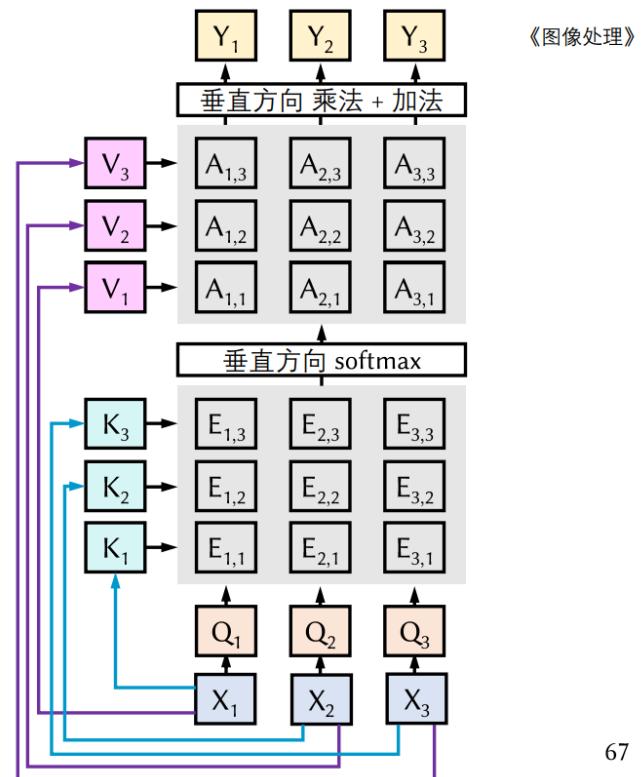
6.3 Transformer

Transformer 的原理不再赘述，此处列出两张图方便回忆：

Transformer

■ 自注意力层

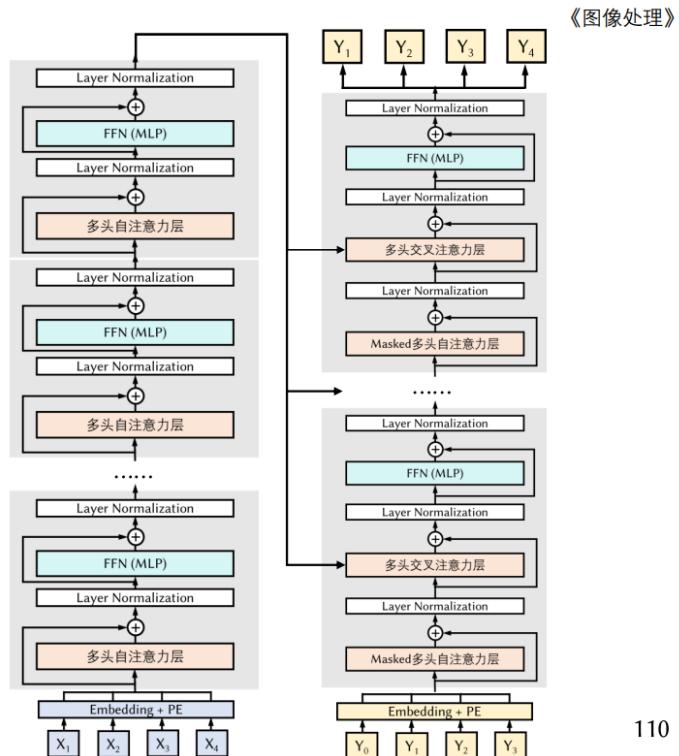
- 输入一组向量 $X (N \times D_x)$
- QKV 分别一个 MLP 层
 - $W_K (D_x \times D_Q), W_Q (D_x \times D_Q), W_V (D_x \times D_V)$
- 计算 Q, K, V
 - $Q = XW_Q (N \times D_Q)$
 - $K = XW_K (N \times D_Q)$
 - $V = XW_V (N \times D_V)$
- 匹配分数 $E = QK^T / \sqrt{D_Q} (N \times N)$
- 权重矩阵 $A = \text{softmax}(E, \text{dim}=1)$
- 输出 $Y = AV (N \times D_V)$



Transformer

■ Transformer

■ 编码器+解码器框架总览



110

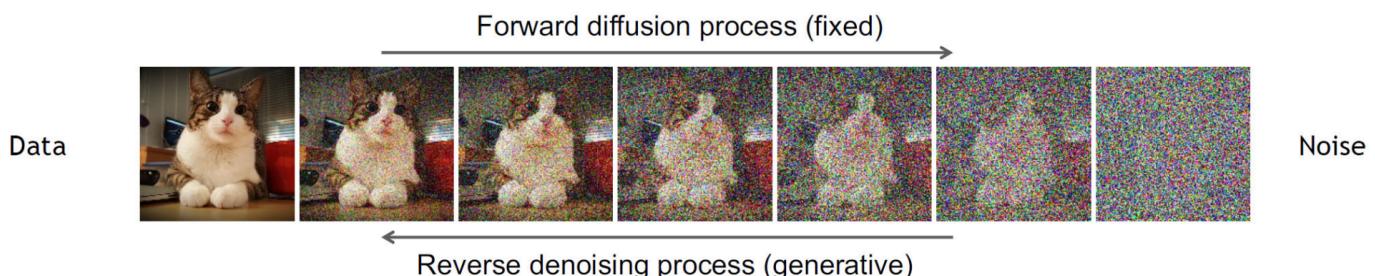
当输入为图像时，则需使用卷积神经网络对图像编码，得到二维特征，然后拉成一维序列再操作。

将自注意力机制与 GAN 结合，可以部分解决卷积层感受野有限，无法建模整体结构的问题，由此搭建的 **SA-GAN** 生成图像性能显著增强。

我们之前还提到了 VQ-VAE，尽管它经过字典映射压缩了图片空间，从而能生成一些高分辨率的图像，但因为解码器质量太差学习效果不佳。为了解决这一问题，后续的工作（例如 **DALL-E**）不再仅依赖更强大的卷积解码器，而是引入 Transformer 对离散潜变量序列进行自回归建模：模型首先使用 VQ-VAE 将图像压缩为离散 token，再利用语言模型式的预测机制生成这些 token 的序列，从而在生成质量与语义一致性上取得显著提升。在此基础上增加 GAN 模块，形成 **VQ-GAN**，可以进一步增加真实性。

6.4 扩散模型 (Diffusion)

扩散模型的过程可以用下图来概括：



其核心思想是，向图像中不断加入高斯噪声，可以逐步使图像变为纯噪声，由此推测可以完成反向过程：从带噪图像预测噪声，从而不断去噪。

首先来看前向过程，设计如下噪声（其中 β_t 为噪声强度）：

$$q(x_t|x_{t-1}) = N(x_t; \sqrt{1 - \beta_t}x_{t-1}, \beta_t I)$$

这是一个技巧性的设计，其目的是令 $\alpha_t = 1 - \beta_t$ ，则重参数化可得到 $x_t = \sqrt{\alpha_t}x_{t-1} + \sqrt{1 - \alpha_t}\epsilon$, $\epsilon \sim N(0, I)$ ，于是逐项代入可以得到一个简洁的式子：

$$x_t = \sqrt{\alpha_t}x_0 + \sqrt{1 - \alpha_t}\epsilon, \quad \bar{\alpha}_t = \prod_{i=1}^t \alpha_i$$

也即 $q(x_t|x_0)$ 也满足高斯分布，我们带着这个结论来看反向过程：我们需要计算 $q(x_{t-1}|x_t) = \frac{q(x_t|x_{t-1})q(x_{t-1})}{q(x_t)}$ ，但其中的 $q(x_{t-1}), q(x_t)$ 均无法计算，也就无法通过模型训练拟合。我们只能另辟蹊径，注意到刚刚我们在前向过程中得出了 $q(x_t|x_0)$ ，于是尝试在训练过程中将拟合目标变为

$$q(x_{t-1}|x_t, x_0) = \frac{q(x_t|x_{t-1}, x_0)q(x_{t-1}|x_0)}{q(x_t|x_0)}$$

代入计算可得（这其中由于前向过程是马尔可夫过程，因此 $q(x_t|x_{t-1}, x_0) = q(x_t|x_{t-1})$ ）真实分布 $q(x_{t-1}|x_t, x_0)$ 也满足高斯分布：

$$q(x_{t-1}|x_t, x_0) = \left(\frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}x_t + \frac{\sqrt{\alpha_{t-1}}\beta_t}{1 - \bar{\alpha}_t}x_0 \right) + \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}\beta_t\epsilon$$

方差部分与参数无关，且可进一步近似为 β_t ，均值部分将前向部分代入写成有关 $x_t, \epsilon_{forward}$ 的函数（注意这里的 $\epsilon_{forward}$ 代表前向过程 x_t 到 x_0 的高斯分布，尽管仍等于 $N(0, I)$ ，但马上就会发现它与另一个 ϵ 的区分必要性）：

$$q(x_{t-1}|x_t, x_0) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}}\epsilon_{forward}\right) + \beta_t\epsilon$$

于是我们可以通过网络进行拟合均值，事实上也就是拟合 $\epsilon_{forward}$ ：

$$p_\theta(x_{t-1}|x_t) = \frac{1}{\sqrt{\bar{\alpha}_t}}\left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) + \beta_t\epsilon$$

直观来看，我们的损失函数也就可以定为 $\|\epsilon - \epsilon_\theta\|$ ，这可以通过与 VAE 类似的变分下界方式严谨推导，但这里篇幅有限，写不下了。

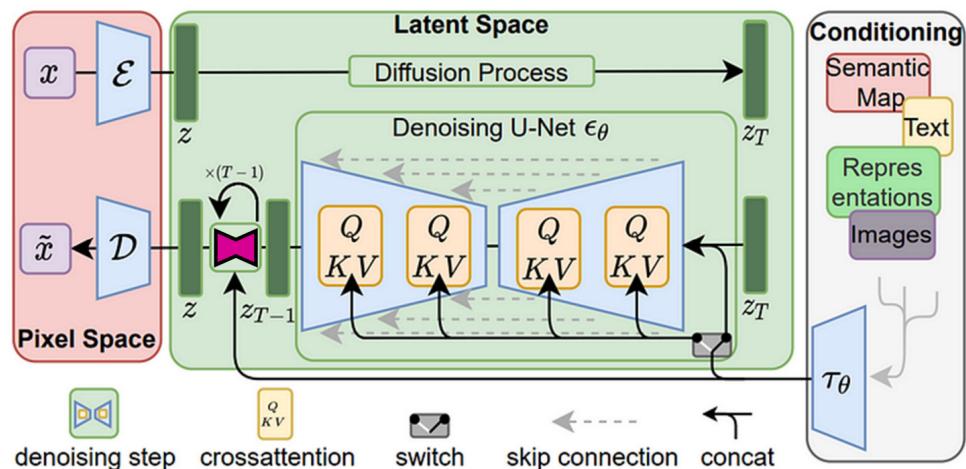
得到这一结果后，训练和采样过程就变得非常显然了（这被称为 **DDPM**）。这样的图像生成效果非常好（与 VAE 对比，其相当于用多步高斯拟合了复杂分布），但是逐步扩散-去噪太慢了，我们希望通过某种意义上的“跳步”实现更少的采样步骤。观察 DDPM 的模式，发现其无法跳步的核心在于其依赖 $q(x_t|x_0, x_{t-1}) = q(x_t|x_{t-1})$ ，而对于 $t \rightarrow t - N$ 的“跳步”采样，这一项无法确定，从而等式左侧的 $q(x_{t-N}|x_0, x_t)$ 的高斯分布形式无法确定（准确地来说，会多一个自由度），但可以通过推导写出它的 μ_t 和 β_t 满足的约束：

$$\mu_t = \sqrt{\bar{\alpha}_{t-N}}x_0 + \sqrt{1 - \bar{\alpha}_{t-N} - \beta_t}\frac{x_t - \sqrt{\bar{\alpha}_t}x_0}{\sqrt{1 - \bar{\alpha}_t}}$$

在实际操作中，我们大道至简地选取 $\beta_t = 0$ ，于是采样过程成为了确定性采样，过程没有了 $t \rightarrow t - 1$ 的约束，可以任意指定 N （当然，仍是步长越小越准确），进行跳步计算，从而加速采样，这便是 DDIM 采样。

Diffusion 在学术界取得了空前的成功，而真正将这一成功转化到产业界的，是基于 Stable Diffusion 的文生图大模型。其设计初衷是基于 Diffusion 存在的一些问题：例如，其直接在图像层面操作，缺乏降维操作来降低算力需求，从而难以应用于高分辨率图像生成；同时，它缺乏输出控制，用户通常需要生成想要的图像，而非随机采样。

Stable Diffusion 在训练阶段和采样阶段分别尝试解决这一问题，在训练阶段，首先使用 VQ-VAE 将像素空间映射为更小的字典空间，再在 Diffusion 去噪模块（U-Net，即下采样-上采样组合）中通过交叉注意力引入 CLIP 将文本提示编码为的视觉特征，具体架构如下图。在采样阶段，可以通过控制超参数 λ 控制输入条件的控制力度，即估计噪声为 $\epsilon'_\theta(x_t, t) = \epsilon_\theta(x_t, t, \emptyset) + \lambda(\epsilon_\theta(x_t, t, c) - \epsilon_\theta(x_t, t, \emptyset))$ ，其中条件 c 可通过 Prompt Engineering 给出。



第七章 渲染

7.1 光照与着色

在渲染过程中，着色无疑是核心环节之一，其根本任务是在特定光照条件下，使物体呈现出某种材质应有的视觉效果。因此，支撑这一过程的关键在于**光源的照明模型与物体表面的反射模型**。

光源通常可以分为几类：环境光（从各个方向均匀照亮场景，用于模拟多次反射所带来的背景亮度）；平行光（如阳光，有方向无衰减）；点光源；聚光源（Spotlight）（如手电筒，有方向有衰减）；面光源（由面积发光的表面组成，计算更复杂，但由于其可产生逼真的柔和阴影与渐变光照因而十分重要）。

接下来介绍常见的反射模型。首先是最典型的**漫反射**。对于粗糙表面，入射光会在微观尺度上向各个方向均匀散射，其亮度满足朗伯定律：

$$I_d = k_d I_l \cos \theta.$$

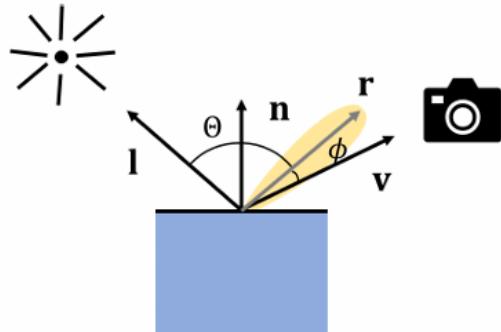
进一步考虑光强随距离的平方反比衰减，并加入恒定的环境光，即可得到最基本的漫反射模型（实际应用中还可能需要考虑多通道颜色计算、远距离的雾化效果等）：

$$I = k_a I_a + f_{att} k_d I_l \cos \theta, \quad f_{att} = \frac{1}{d^2}.$$

另一类重要的反射现象是**镜面反射**，其亮度取决于观察方向与理想反射方向之间的夹角衰减，可用 Phong 提出的经验模型描述：

$$I_s = k_s I_l (\cos \phi)^{n_s},$$

其中 ϕ 为**视线方向与反射方向之间的夹角**（如下图），指数 n_s 决定高光的锐利程度： n_s 越大，高光越集中、边缘越尖锐。



综合上述漫反射与镜面反射项，得到经典的 **Phong 光照模型**：

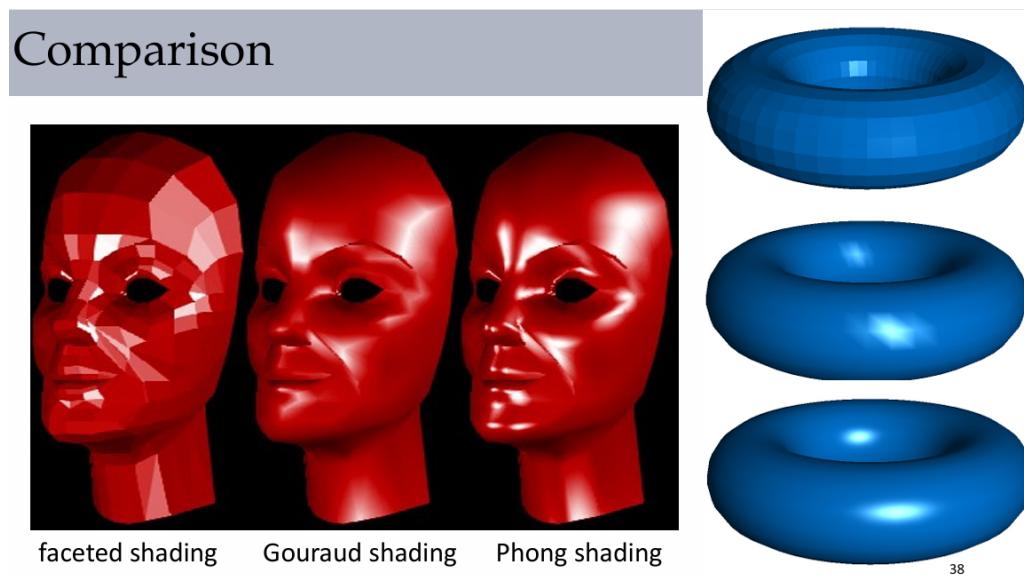
$$I = k_a I_a + f_{\text{att}} I_l (k_d \max(0, \cos \theta) + k_s (\cos \phi)^{n_s}) .$$

其中 $\max(0, \cos \theta)$ 用于保证漫反射仅在光线入射角为锐角时才产生贡献（即光源位于表面朝向的一侧时才有有效的入射光）。

此前讨论了如何在单个点上执行光照计算，接下来需要将这一过程推广到整个几何体的着色。最朴素的方法是**平面着色 (Flat Shading)**。该方法以三角形或四边形网格为基本单元，在每个单元的重心处只计算一次光照，并将该结果填充至整个面片。虽然实现简单，但相邻面片之间缺乏连续性，产生明显的“分面感”。

一种改进方法是对网格内部的颜色进行插值，从而得到更加平滑的结果，即**Gouraud 着色**。在表面变化较为平缓时，Gouraud 着色能够获得相当不错的效果。然而，该方法的原理事实上并不严格：它在屏幕空间 (screen space) 而非世界空间 (world space) 执行线性插值。由于投影变换本身是非线性的，在投影后的屏幕空间中进行插值无法保持表面空间中的线性关系（尽管在大多数情况下这一误差不明显）。此外，由于 Gouraud 着色的结果仍然与几何形状关系不强，因此仍可能出现“面状”伪影。例如，当高光区域并未落在任何顶点上时，单纯的线性插值将无法正确重建高光。

Phong 着色进一步提升了这一过程：它在顶点之间线性插值的是**法线**而非颜色，从而可以在每个像素处更精确地计算光照模型，因而能够更准确地反映表面的真实光照变化。以下是三种着色模型在同一例子上的表现：



在此前的讨论中，我们理想化地忽略了**阴影**的处理。然而在决定光照强度之前，首先必须解决物体之间的**可见性**问题。

最朴素的可见性算法是**画家算法**：按照深度从后往前依次绘制物体。但这种方法依赖对所有物体进行排序，而排序本身既昂贵又常常不可行——例如物体之间发生互相穿插时，甚至需要对几何体进行复杂的分割。

为此，现代渲染通常采用逐像素的 Z-buffer 技术。其核心思想是：在扫描所有物体时，为每个像素维护当前离视点最近的深度值 $zbuf[x, y]$ 。若新片段的深度更近，则更新该像素的深度与颜色。Z-buffer 方法简单高效，但也存在局限：它难以处理透明物体，并且在深度精度不足时会产生排序伪影。此外，当场景包含大量面片时，逐片段测试的成本很高，因此通常需要结合空间加速结构（如 BSP Tree、Octree 等）进行优化。一个最简单而常用的优化是背面剔除（Backface Culling）：对于背对视点的多边形（满足 $V \cdot N \geq 0$ ），直接跳过渲染，可使面片数量减少约一半，从而明显提升效率。

在上述可见性判断机制的基础上，我们可以进一步扩展之前的着色算法，构建 Shadow Map：从光源视角生成一张记录“最靠近光源的深度”的缓冲区。渲染时将像素位置变换至光源空间，并与 Shadow Map 中的深度进行比较，从而判断该点是否被遮挡，从而实现阴影的正确呈现。

除了能够实现高效的真实感渲染之外，光栅化的另一个重要应用方向是非真实感渲染（NPR）。NPR 关注的重点并非物理真实性，而是如何通过“线条与着色”更好地表达结构与风格。

在线条表达方面，人们往往希望通过轮廓线来突出物体的几何特征。常见的方法之一是基于法线与视线方向的夹角：当该角度接近 90° 时，将其判定为轮廓，但仅控制 $n \cdot v \approx 0$ 的精度不易保证线条的等宽性。另一类方法则通过深度偏移实现描边：不删除背面，而是将背面的 z 值略微减小后重新渲染，使其在正面的几何之外“露出”一圈，从而自然形成轮廓描边；也可以结合深度学习的边缘提取方法获得更复杂的边线效果。

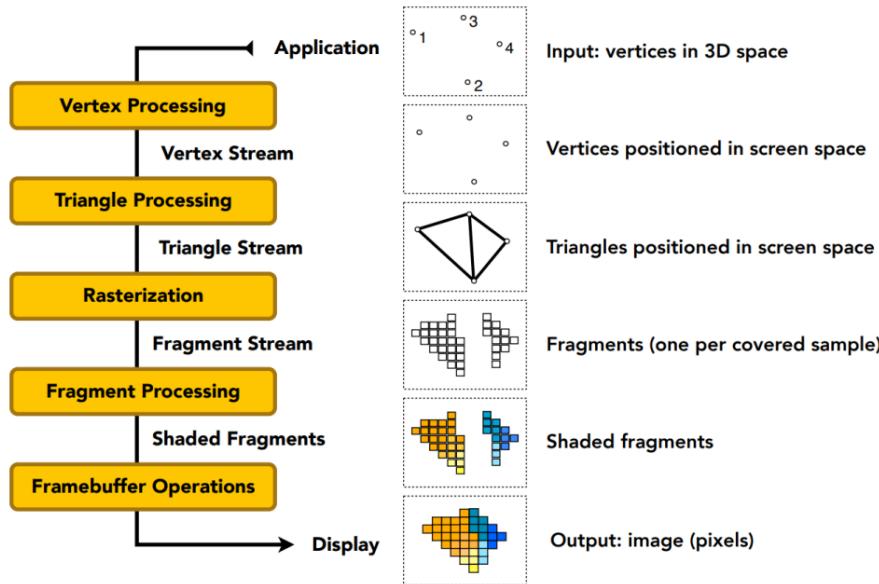
在着色方面，一个经典的 NPR 技法是冷暖着色（Gooch Shading）。它以冷色与暖色替代传统的亮暗关系：

$$I = \left(\frac{1 + n \cdot l}{2} \right) k_{\text{cool}} + \left(\frac{1 - n \cdot l}{2} \right) k_{\text{warm}}$$

这种方法弱化了真实光照的强对比，使物体各部分都能被更清晰地识别，从而更强调形体结构而非真实光照效果。

7.2 渲染管线

上一节介绍了光栅化的基本原理，本节将系统性地梳理其在实际中的应用，即光栅化渲染的整体管线。这一流程可以用下图直观概括：



其主要阶段包括：**顶点处理**（对顶点进行一系列几何变换，计算其在屏幕空间中的位置）；**三角化**（将顶点组装为三角形，并执行剔除与裁剪操作）；**光栅化**（确定三角形覆盖的片元 fragment【注意其不同于最终像素，它包含深度信息】并对顶点属性做插值）；**片元着色**（根据上一节介绍的光照模型进行着色，并处理纹理贴图）；**最终输出**（进行 Z-buffer 测试，写入颜色缓冲区，生成最终像素）。

在 GLSL 着色器中，主要可编程阶段为顶点着色器（逐顶点计算位置并输出用于插值的属性）与片元着色器（逐片元接收插值后的属性并输出颜色），二者共同定义了渲染效果。

然而，最朴素的**前向渲染**具有 $O(NM)$ 的时间复杂度，其中大量片元着色会被后续遮挡而被丢弃，造成显著浪费。为提升效率，可采用两类优化策略：其一是**提前深度测试**（Early Z），即在执行着色前利用 Z-buffer 判断片元是否被遮挡，从而跳过不必要的着色计算；其二是在遍历三角形时仅将几何属性写入 G-Buffer，而将光照计算推迟到屏幕空间中逐像素完成，使光照复杂度与三角形数量脱钩，从根本上减少冗余开销，这一策略即**延迟渲染**（Deferred Rendering）。

7.3 纹理映射

本节是经典光照-着色流程的补充。在经典光照模型中，像素的最终亮度通常由环境光、漫反射和高光三部分组成，可表示为：

$$I = k_a I_a + f_{att} I_l (k_d \cos \theta + k_s (\cos \phi)^{n_s})$$

然而，如果希望场景中不同位置的表面在颜色、粗糙度或反射特性上呈现差异，仅依赖固定参数显然是不够的。为了使这些参数能够随空间位置变化，引入了**纹理映射**的思想。

纹理映射的核心目标是：以世界坐标系为中介，将**屏幕空间中的像素坐标**映射到**纹理空间的 (u, v) 坐标**，从而为每个片元分配来自二维纹理图像的属性值。借此，原本光滑、单调、仅具几何形状信息的模型便能“贴上皮肤”，获得丰富的表面细节与更加真实的视觉效果。

在渲染管线中，这一过程属于片元处理阶段。然而，如果在片元阶段直接对屏幕空间中的属性进行线性插值，就会产生显著的透视畸变——因为屏幕空间与世界空间之间并不存在线性映射关系。

为避免这一问题，需要采用透视校正插值。其基本思路是：在插值时依据顶点的深度（齐次坐标中的 w 分量）对属性进行加权，从而补偿透视投影带来的非线性失真。其公式如下：

$$f = \frac{\frac{\alpha}{w_A} f_A + \frac{\beta}{w_B} f_B + \frac{\gamma}{w_C} f_C}{\frac{\alpha}{w_A} + \frac{\beta}{w_B} + \frac{\gamma}{w_C}}$$

其中， f_A, f_B, f_C 为顶点属性（如纹理坐标、颜色或法线）， w_A, w_B, w_C 为各顶点透视投影前的深度分量。

进一步地，对深度值本身而言：

$$\frac{1}{w_P} = \frac{\alpha}{w_A} + \frac{\beta}{w_B} + \frac{\gamma}{w_C}$$

由此可见，深度倒数在屏幕空间中是线性变化的，这正是透视校正插值成立的数学基础。

而透视投影又会导致远处物体在屏幕上对应过大的纹理区域，从而产生摩尔纹等明显的反走样问题；此时即可使用前面介绍的 MIPMAP 等反走样技术来有效缓解。

纹理映射在现代图形学中具有多种用途：它可以为物体表面赋予真实的颜色与材质属性（如粗糙度、金属度等）；也可以通过凹凸贴图、法线贴图等方式模拟细致的几何结构；此外，还能用于全局光照与阴影计算，如阴影贴图（Shadow Map）等方法。

最后简单讨论纹理贴图的获取与生成。传统且最常用的方式是通过实拍采集，但其挑战在于如何处理重复使用时的边界过渡问题（否则会出现明显的拼接痕迹）。可以采用简单的图像翻转来平滑边缘，也可以借助更高级的纹理合成技术（相关内容将在其他章节讨论）。此外，程序化纹理生成与基于 AI 的纹理合成也是常用的方案。

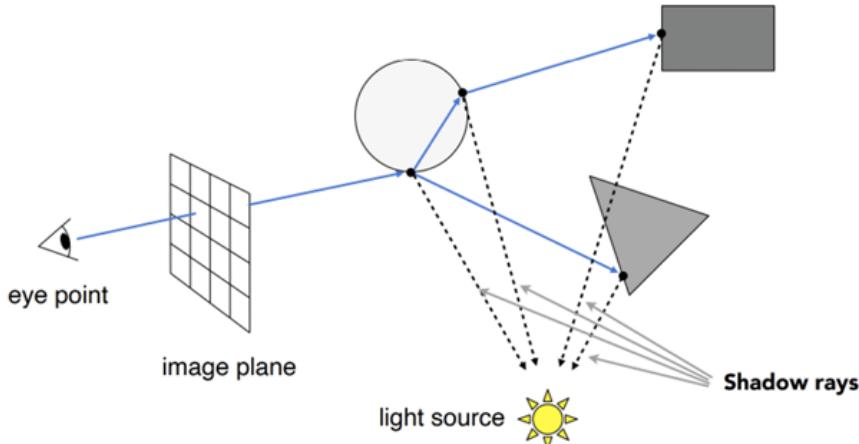
7.4 光线追踪

尽管光栅化管线已相当成熟，但其在表达全局光照方面仍存在局限，因此高级渲染（如光线追踪）逐渐成为补充与发展方向。本节便介绍光线追踪相关。

首先介绍**光线投射（Ray Casting）**，这是光线追踪的前身，其核心思路是将计算过程从对每个物体的遍历转变为对每个像素的遍历。具体而言，它从视点出发，为图像中的每个像素发射一条光线（Eye Ray），并追踪其在场景中的路径。每条光线依次与所有物体表面进行相交测试（这一过程最为耗时），找到距离视点最近的交点后，根据该点的法线、材质与光照信息执行着色计算，从而确定该像素的最终颜色。

光线投射在机制上对光栅化进行了优化，其可以处理一些半透明的情况（一个像素沿视线方向可以同时看到多个深度上的贡献，这是 Z-buffer 机制天生做不到的）。更重要的是，它可以自然地推广为循环的**光线追踪（Whitted-Style Ray Tracing）**，其流程即为：

对于每个像素发射视线光线，寻找最近的交点，进而从该交点向每个光源发出一条阴影光线（Shadow Ray），判断路径中是否有遮挡，以判断该点是否被照亮；若其被照亮，便可以根据 Phong 光照模型等计算。若表面具有镜面或透明特性，则需进一步沿反射方向生成反射射线（Reflection Ray）或沿折射方向生成折射射线（Transmission Ray），并重复上述步骤，即完成光线追踪。尽管这已经非常清晰了，还是给一张示意图：



以下详细讲一下求交的过程。一条射线可以由其起点和方向表示为 $x = \mathbf{p} + t\mathbf{d}$ ，事实上求交就是将这一方程代入目标物体的方程进而求解参数 t 。

若目标物体使用隐式方程 $f(x) = 0$ 表示的，代入即得 $f(\mathbf{p} + t\mathbf{d}) = 0$ ，一个等式一个未知数 (t)，从而是可解的，例如对球面，代入后得到二次方程

$$(p_x + td_x)^2 + (p_y + td_y)^2 + (p_z + td_z)^2 = r^2$$

展开后求解关于 t 的二次方程即可（取结果中较小的正值）。

而对于显式方程（如三角形）可以先求解与平面方程 $(\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} = 0$ 的交点（这就转化为了一个隐式的问题），再判断交点是否在多边形内部即可（对于三角形，这一步可以用重心坐标 $p = b_1x_1 + b_2x_2 + b_3x_3$ ，判断 $b_1, b_2, b_3 > 0$ 即可；对于多边形则更复杂，可能需要三角化）。

对于远处物体，光线追踪结果仍可能出现走样，此时可采用前文介绍的超采样反走样技术加以处理。

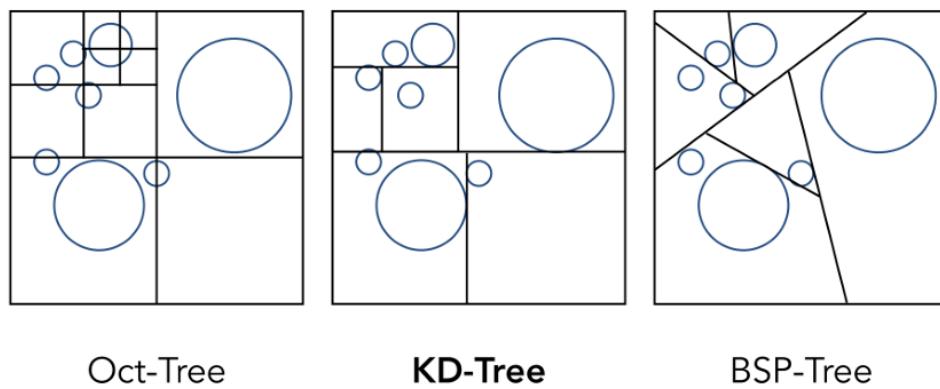
在渲染高速运动的物体时，由于曝光时间对应的是一个时间区间，物体的不同部分会在这一时间段中经历多个位置。若希望模拟真实世界中的运动模糊效果，需要在时间维度上累积多个时刻的物体状态，实现一种基于视觉暂留的反走样，这一技术即 Motion Blur。

以上介绍了最基础的渲染算法，接下来的任务便是对其进行优化加速。光线追踪的主要性能瓶颈集中在两方面：其一，光线需要与场景中每个物体进行求交测试；其二，许多物体本身具有复杂几何形状，使得单次求交计算就已经十分昂贵。针对这两个问题，人们提出了一系列空间加速结构，用于显著减少光线求交次数。

网格加速 (Grid Acceleration) 的核心思想是：不再让光线逐一与所有物体求交，而是只与光线穿过的**少量网格单元**中的物体求交。基本流程为：首先为整个场景建立**包围盒**；将空间划分为规则的网格；将每个物体登记到其覆盖的格子中；光线从场景入口出发，仅在当前格子中测试相交物体，然后沿着光线方向跳转到下一个格子。由于光线只需访问有限数量的格子，计算量大幅降低。

然而，规则网格存在天然的缺陷：格子大小难以选择。若格子过大，许多物体集中在少数格子内，几乎没有加速效果；若格子过小，光线跨越格子的次数过多，带来额外的存储和遍历成本。因此，人们进一步发展出更灵活且自适应的**层次化空间划分结构**，根据物体分布递归划分空间，得到层次加速结构。对每条光线，只需从根节点沿光线路径递归访问可能被穿过的节点，并仅在叶子节点中进行精确求交测试。

最典型的层次加速结构是**八叉树 (Octree)**：将空间递归划分为 8 个子立方体，并仅在物体密集区域继续细分，从而节省存储并加速查询。**K-d Tree** 则在每次分割中仅选择一个坐标轴，并按该轴上的中位数进行划分，使树结构更为平衡；**BSP Tree** 更加自由，直接使用任意方向的超平面进行切分，常用于画家算法中按“从后往前”的顺序遍历空间。其示意图如下：



7.5 路径追踪

Whitted-Style Ray Tracing 虽然能够处理多次反射，但其反射模型主要局限于理想的镜面反射与漫反射，难以刻画现实中粗糙表面的复杂反射行为。因此，我们需要更通用、更物理的渲染方法。本节将介绍一种基于能量守恒的光线追踪方法——**路径追踪 (Path Tracing)**。其核心思想是：光线在物体表面发生反射或折射时，其能量传输过程必须满足物理上的守恒规律。为了精确描述这种能量关系，我们需要引入一套更严格的物理语言——**辐射度量学**。

我们首先引入电磁辐射中的“功率”——**通量** Φ ，在此基础上，引入两个（在本节中）重要的物理量：**辐照度** $E(p) = \frac{d\Phi(p)}{dA}$ ，其描述了单个点 P 单位时间内收到的能量；以及**辐射率** $L(p, \omega) = \frac{dE_\omega(p)}{d\omega}$ ，其中 E_ω 代表的是垂直于 ω 方向的表面的辐照度。

由某一表面的一组入射光和反射光的强度成比例这一自然的假设，定义二者比例 f_r ，这便是

描述表面材质反射模型的物理量——双向反射分布函数 (BRDF)。由上述定义，

$$f_r(p, \omega_i, \omega_r) = \frac{dL_o(p, \omega_o)}{dE(p, \omega_i)} \text{(想象增加一束光的过程)}$$

又由 $L_i(p, \omega_i) \cos \theta = \frac{dE(p, \omega_i)}{d\omega}$ (定义)，代入可得：

$$f_r(p, \omega_i, \omega_r) = \frac{dL_o(p, \omega_o)}{L_i(p, \omega_i) \cos \theta d\omega_i}$$

于是我们对每个方向的出射光累计各束入射光的贡献，得到

$$L_o(p, \omega_o) = \int_{\Omega} f_r(p, \omega_i, \omega_r) L_i(p, \omega_i) \cos \theta d\omega_i$$

其中的 f_r 这一反射系数可以自然地推广到折射中。最后，若其本身为点光源，则贡献还要累加上本身的发光强度 $L_e(p, \omega_o)$ 。这便得到了最终的光传输方程 (LTE)：

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{\Omega} f_r(p, \omega_i, \omega_r) L_i(p, \omega_i) (\mathbf{n} \cdot \omega_i) d\omega_i$$

这显然无法显式求得解析解，只能借助数值方法进行逼近。在路径追踪中，我们采用蒙特卡罗积分对渲染方程进行估计：将原本的积分离散为随机采样的求和。具体而言，若在半球上根据概率分布 $X_i \sim p(x)$ (可为均匀分布) 采样方向 ω_i ，则有

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \frac{1}{N} \sum_{i=1}^N \frac{f_r(p, \omega_i, \omega_o) L_i(p, \omega_i) (\mathbf{n} \cdot \omega_i)}{p(\omega_i)}.$$

将其写入算法即形成类似光线追踪的多重递归结构。然而这一实现仍面临两个问题：光线反弹在理论上没有自然终止条件；并且随着反射次数的增加，采样数量呈指数级增长。为此，原论文引入了极为巧妙的**俄罗斯轮盘赌策略**，在保持无偏性的前提下有效地截断路径。其做法是：每次反射时，以概率 P 决定是否继续递归追踪光线；若终止则直接返回 0，否则继续，但需要将该路径的贡献除以 P 以放大权重，从而保证整体估计仍保持无偏。

这便完成了整个路径追踪的过程。我们指出，尽管这是一个随机算法，但其效果已经完整贴合真实场景。还有一些小注意点：若每个像素时进行一次采样可能会有较多噪声，可以在像素区域内取多个点进行采样，使结果更平滑；这一超参 (SPP) 与反弹次数的增大都会使得渲染效果显著变好，但会存在与效率的 trade-off。

第八章 仿真

8.1 物理模拟

动画与仿真的生成方式大体可分为四类：人工操作、程序化方法、物理驱动以及数据驱动。其中，**动画（Animation）** 更强调对动作效果的掌控，因此往往依赖数据驱动技术；而**物理模拟（Simulation）** 旨在真实再现物理现象，因而更多地依赖物理规律来驱动系统的演化。

对于单个粒子而言，物理仿真相对简单：只需根据受力进行时间积分，便能得到速度与位置随时间的变化。然而，现实世界中的物体由无数粒子构成，粒子之间存在复杂的耦合与相互作用，使得系统难以获得解析解。因此，我们通常只能采用**数值方法**来近似求解其运动行为。

这样的数值物理模拟系统通常包含三个核心部分：**Kinematics**（运动学，描述位置的变化率等于速度）、**Dynamics**（动力学，描述速度的变化率由受力决定）以及 **Approximation**（近似，用以在计算机中表示连续系统）。由此引出了数值模拟中的三项基本技术：**时间离散化、空间离散化与数值求解**，它们共同构成从物理定律到可计算模型的实现路径。

于是，我们需要进一步回答三个关键问题：空间如何表示？动力学方程如何离散？以及，如何在时间上进行积分？

空间离散化的方法如下：将连续的物体用有限数量的粒子进行近似表示，并通过**弹簧系统**将这些粒子“彼此相连”，以此模拟材料的弹性与柔性特征。对于连接粒子 i 与 j 的弹簧，定义 $\mathbf{x}_{ij} = \mathbf{x}_j - \mathbf{x}_i$ ，则该弹簧的势能为

$$E_{ij} = \frac{1}{2}k (\|\mathbf{x}_{ij}\| - l_0)^2,$$

根据胡克定律，粒子 i 受到来自粒子 j 的弹力为

$$\mathbf{f}_{ij} = k (\|\mathbf{x}_{ij}\| - l_0) \frac{\mathbf{x}_{ij}}{\|\mathbf{x}_{ij}\|} = -\mathbf{f}_{ji}.$$

最终，粒子 i 所受的合力可写为

$$\mathbf{f}_i = \sum_{j \in N(i)} \mathbf{f}_{ij} + \mathbf{f}_i^{\text{ext}}(\text{外力项})$$

接下来就是运动学的范畴，我们首先尝试将积分转化为离散形式，即使用上一步的速度、力更新整体的下一步的速度、位置：

$$x(t_n) - x(t_{n-1}) \approx v(t_{n-1})\Delta t, \quad v(t_n) - v(t_{n-1}) = \frac{1}{m} f(t_{n-1})\Delta t$$

但我们指出，无论时间步/网格多小，理论上都一定会导致数值爆炸（无条件不稳定），其理论解释可以从简单的无阻尼弹簧系统的粒子看出：

因此，需要引入更稳定的时间积分方法来提高系统的数值稳定性。隐式欧拉积分的形式为：

$$x(t_{n+1}) = x(t_n) + v(t_{n+1})\Delta t, \quad v(t_{n+1}) = v(t_n) + \frac{1}{m}f(t_{n+1})\Delta t.$$

该方法虽然仍为一阶精度，但相比显式欧拉具有显著更高的稳定性。具体可写为：

$$\begin{aligned} x_{n+1} &= x_n + hv_n + h^2M^{-1}f(x_{n+1}) \\ &= (x_n + h(v_n + hM^{-1}f^{ext})) + h^2M^{-1}f^{int}(x_{n+1}) \text{ (分离外力和内力)} \\ &= y + h^2M^{-1}f^{int}(x_{n+1}), \end{aligned}$$

其中 y 表示与当前步内力无关的项，即惯性项，这就得到了一个只与内力有关的方程。进一步地，我们可以将该更新过程转化为一个最优化问题：

$$\begin{aligned} \nabla g(x) &= \frac{M}{h^2}(x - y) - f^{int}(x) = 0 \\ \Rightarrow x_{n+1} &= \arg \min_x g(x), \quad g(x) = \frac{1}{2h^2}\|x - y\|_M^2 + E(x), \quad \text{其中} \|x - y\|_M^2 = (x - y)^T M(x - y) \end{aligned}$$

其中第一项代表“惯性趋势”，第二项为系统的势能。通过在每个时间步最小化该能量函数，实现了惯性与势能之间的平衡，从而保证了在任意时间步长下的稳定性。

在求解角度，可以使用牛顿迭代进行优化：给定初始迭代点 x_1 ，不断更新 $x_{k+1} = x_k - H(g)^{-1}\nabla g$ （其实即相当于将梯度下降的各方向步长人为设定为 $H(g)^{-1}$ ）。

然而，牛顿步可能会“走得过远”，导致能量不降反升。为此通常会结合 **线搜索 (Line Search)** 来增强稳定性。具体做法是设方向 $p_k = -H(g)^{-1}\nabla g$ ，以 $\alpha = 1$ 为初值，不断更新 $\alpha \leftarrow \alpha/2$ ，直到满足 $g(x_k + \alpha p_k)$ 相较于 $g(x_k)$ 实际下降为止。

我们指出，如果能够进行较为合理的人为参数设定，弹簧质点系统是一种结构清晰且稳定性良好的物理建模方式。若希望获得更高的物理精确度，可以采用**有限元方法**，它通过将连续体划分为有限数量的单元并在局部求解后整体拼接来逼近连续场的行为，本课程中不作深入介绍。

接下来是流体动力学，其原理更复杂，但其本质仍是 $ma = F$ ，只是换成了“升级版”的公式——**Navier-Stokes 方程**：

$$\rho \frac{D\mathbf{v}}{Dt} = -\nabla p + \rho g + \mu \nabla^2 \mathbf{v}$$

等式左侧是密度乘以流体粒子本身的加速度，相当于 ma ，是在粒子视角即拉格朗日视角下的表达式，右侧三项分别代表压力、重力、粘性力（在物理模拟中常常可忽略），相当于 F ，是针对空间场，在网格视角即欧拉视角下的表达式。这一方程事实上完成了两个视角的统一，即每个流体粒子的加速度，等于它所在空间点的力场对它的作用。

需要注意的是， $\frac{D\mathbf{v}}{Dt}$ 与 $\frac{\partial \mathbf{v}}{\partial t}$ 并不相同，前者代表在某个粒子视角其加速度，后者代表空间中某个固定点处流体的加速度，二者关系为：

$$\frac{D\mathbf{v}}{Dt} = \frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla) \mathbf{v},$$

其中第二项为**对流项**，反映了流体粒子因在速度场中移动而产生的额外速度变化。

可以看出在 N-S 方程中，未知量是流体速度和空间压强。这个问题事实上是欠定的问题，我们需要引入第二个约束——**质量守恒方程**：

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho v) \quad (\text{流出密度})$$

而在流体仿真（尤其是水的仿真）中，我们可以假设密度为常数，因此可以直接得到

$$\nabla \cdot v = 0$$

这被称为流体的**不可压缩性**。

接下来介绍具体如何求解流体系统。从粒子角度，我们可以存储单个粒子的质量、速度、位置等信息，但是我们仍然需要**连续的密度场**来进一步得到压力梯度等信息。连续密度场可由周围粒子的“按距离平滑加权”得到。一种常用的距离加权核函数是：

$$W(r) = \frac{315}{64\pi d^9} \begin{cases} (d^2 - r^2)^3 & 0 \leq r \leq d \\ 0 & \text{otherwise} \end{cases}$$

这是一个归一化的权重函数，于是便可以得到密度场的表达式：

$$\rho(x) = \sum m_i W(|x - x_i|)$$

进而也可以求得压力场（在多方过程中， $p \propto \rho^\gamma$ ，因此单个粒子周围的压力为 $p_i = k(\rho_i - \rho_0)^\gamma$ ）：

$$p(x) = \sum p_i \frac{m_i}{\rho_i} W(|x - x_i|)$$

加权系数是由于压力是定义在体积上的场量，可以通过量纲理解。

具体到算法层面，步骤为：先利用重力加速度 g 更新速度和位置；再估计密度场，计算压力，进而利用压力 $-\frac{m \nabla p}{\rho}$ 更新速度与位置即可，这就是经典的**SPH 算法**。压力的表达式可以根据上面的推导求得：

$$f = -\frac{m \nabla p}{\rho} = -\frac{m_i}{\rho_i} \sum_j p_j \frac{m_j}{\rho_j} \nabla_i W(|x_j - x_i|)$$