CPT205W11_裁剪 Clipping

剪切和栅格化 Clipping & rasterization

剪切（clipping）：删除剪切窗口（clipping window）之外的对象或对象的一部分。

光栅化 （Rasterization/Scan conversion扫描转换）：在帧缓冲中将高级对象描述转换为像素颜色。

图形系统（如 OpenGL）为我们完成了这些工作：不需要明确的 OpenGL 函数来进行剪切和栅格化。

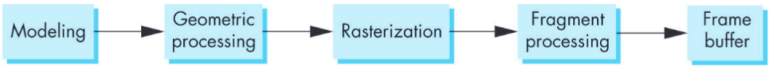
为什么要剪切?

栅格化开销极大，几乎正比于创建的片段（fragments(candidate pixels)），同时要计算每个像素的数学与逻辑。

如果只栅格化实际可见的内容，可以节省大量计算

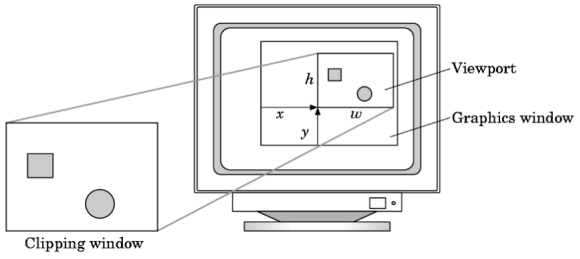
所需任务：

- 剪切Clipping
- 变换Transformations
- 栅格化Rasterization
- 推迟到片段处理（fragment processing）的任务：去除隐藏表面（Hidden surface removal），抗锯齿（Antialiasing）



剪切窗口与视口（viewport）：剪切窗口选择我们想在虚拟 2D 世界中看到的内容。视口指示在输出设备 （或显示窗口内） 查看数据的位置。默认情况下，视口的位置和尺寸与我们创建的 GLUT 显示窗口相同。

```
void glViewport(GLint x, GLint y, GLsizei w, GLsizei h)
```



视口是状态的一部分，

剪切基元 Clipping primitives

不同的基元（点，线，多边形）可以用不同的方式处理

2D objects -> clipping window

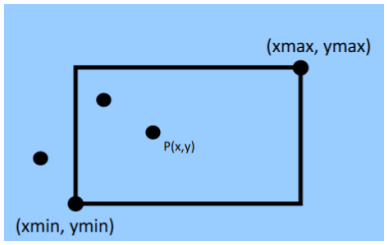
3D objects -> clipping volume

易于用于线段和多边形，难以用于曲线和文本（首先转换为线条和多边形）

2D剪切

点

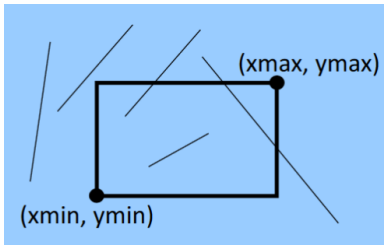
确定点 （x，y） 是在窗口内部还是外部。




```
if (xmin <= x <= xmax) and (ymin <= y <= ymax)
    the point (x,y) is inside
else
    the point (x,y) is outside
```

线

确定一条线是在窗口内部、外部还是部分在窗口内部。如果一条线部分在内部，我们需要显示内部的部分。



分类讨论：

平凡情况：

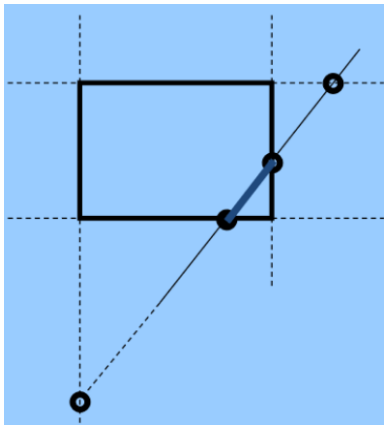
Trivial accept：线完全在剪切窗口内

Trivial reject：线段完全在剪切窗口外

非平凡情况：

一点内，一点外。

两点外，但有部分在窗口内，需要找到内部的线段



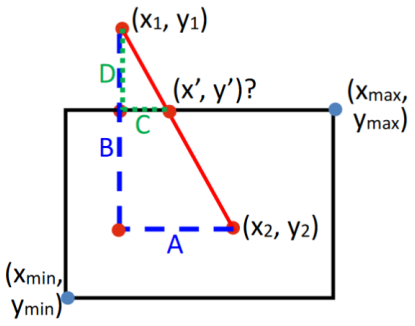
计算线与窗口边界的交界点，两个点就是线段的终点。

蛮力裁剪（Brute force clipping of lines）

使用y=mx+b求解联立方程，但无法处理垂线

Similar triangles

A / B = C / D
A = (x_2 - x_1)
B = (y_2 - y_1)
C = (x' - x_1)
D = (y' - y_1) = (y_{max} - y_1)
➞ C = A•D / B
➞ x' = x_1 + C
➞ (x', y') = (x_1 + C, y_{max})



但开销太大，对2d需要4个浮点减法，一个浮点乘法，一个浮点除法，重复4次

Cohen-Sutherland

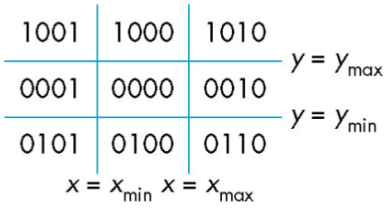
2D的情况

为每个断点创建一个编码（outcode），包含该点相对剪切窗口的位置信息。使用两个端点的编码确定线条相对剪切窗口的配置，如有必要计算新的端点，这可以轻松扩展到3d（使用6 个面而不是 4 个边）

- Split plane into 9 regions.
- Assign each a 4-bit outcode (left, right, below, and above).

b_0 = 1 if $x < x_{min}$, 0 otherwise
b_1 = 1 if $x > x_{max}$, 0 otherwise
b_2 = 1 if $y < y_{min}$, 0 otherwise
b_3 = 1 if $y > y_{max}$, 0 otherwise

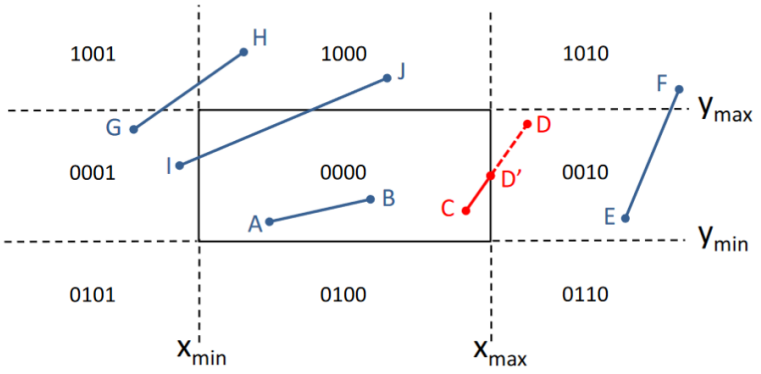
- Assign each endpoint an outcode.



如图将平面分为9个部分，每个部分分配4-bit outcode（前2bit代表10上00中01下，后2bit代表10右00中01左）

计算最多需要4次减法

- CD (one endpoint inside and one outside clipping window)
  - outcode (C) = 0000, outcode(D) = 0010 ≠ 0000
  - Compute intersection
  - Location of 1 in outcode(D) determines which edge to intersect with (right edge in this case)

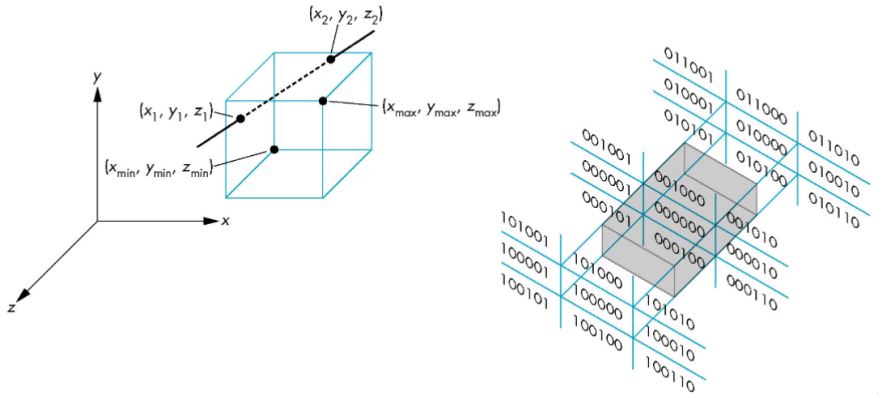


- AB：两端点均在剪切窗内 A=B=0000 平凡接受（Trivial Accept）
- CD：一端点内一端点外 C=0000!=D=0010 计算相交（intersection）D编码中1的位置决定了与哪条边相交。
- EF：两端点均在剪切窗口外 将两端点代码取与E0010 && F1010 = 0010 !=0000 说明线段全处于窗口外，平凡拒绝
- GH和IJ：两端点均在剪切窗口外 取与=0000 需要找到任意一个线段与边缘相交的点，确定交点的编码来决定是否需要绘制

在许多应用中，剪切窗口对于整个场景很小，大多数线段位于窗口之外或一侧，可以直接通过outcode消除。当必须缩短线段计算交点时需要额外步骤，不高效。

3D的情况

使用6位outcode，6个剪切面，27个区域。



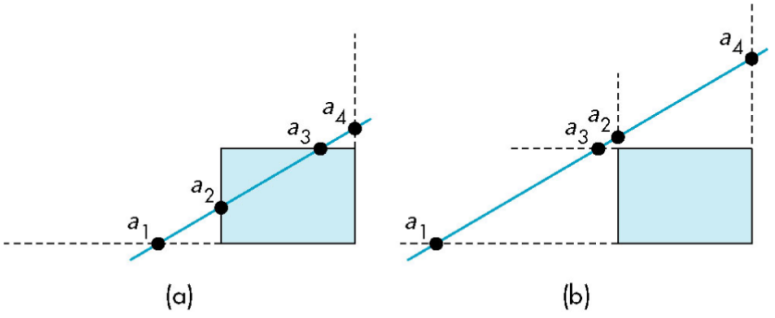
Liang-Barsky

考虑线段的参数形式：$p(a) = (1-a)p_1 + ap_2 \quad (0 \le a \le 1)$

或：$x(a) = (1-a)x_1 + ax_2 \quad y(a) = (1-a)y_1 + ay_2$

我们可以通过a的不同区分线段哪部分在窗口内

- In (a): $\alpha_4 > \alpha_3 > \alpha_2 > \alpha_1$
  Intersect right, top, left and bottom: shorten
- In (b): $\alpha_4 > \alpha_2 > \alpha_3 > \alpha_1$
  Intersect right, left, top and bottom: reject



优点：可以像Cohen-Sutherland一样容易计算接受或拒绝，使用a的值避免了CS法的递归使用，容易扩展到3D

平面-线相交

用矩阵形式写出线和平面的方程：（n是平面法线，P_0是平面上一点）

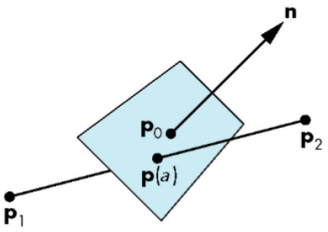
即解方程：$p(a) = (1-a)p_1 + ap_2 \ \ (0 \le a \le 1)$和$n \cdot (p(a) - p_0) = 0$

对于与交点对应的a，值为：

For the α corresponding to the point of intersection, this value is

$$\alpha = \frac{\mathbf{n} \bullet (\mathbf{p}_o - \mathbf{p}_1)}{\mathbf{n} \bullet (\mathbf{p}_2 - \mathbf{p}_1)}$$

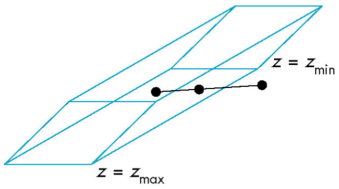
The intersection requires 6 multiplications and 1 division.



需要计算6次乘法1次除法

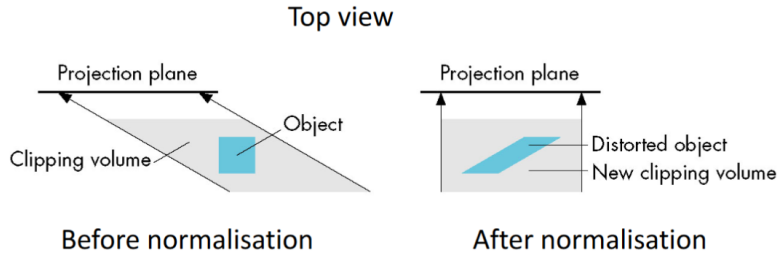
General 3D clipping

计算线段与任意平面相交（如斜视图oblique view的情况）



归一化形式（normalized form）是viewing的一部分，但在归一化后我们贴着平行六面体的侧面进行剪辑。

使用浮点乘法进行相交计算。

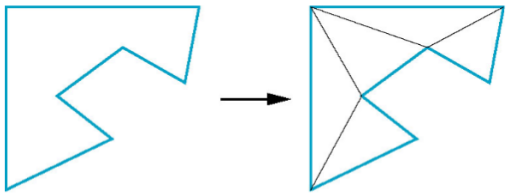


多边形裁剪

并不像线段裁剪，因为裁剪线段最多生成一个线段，裁剪多边形可能生成多个多边形（若凸多边形则最多一个）。

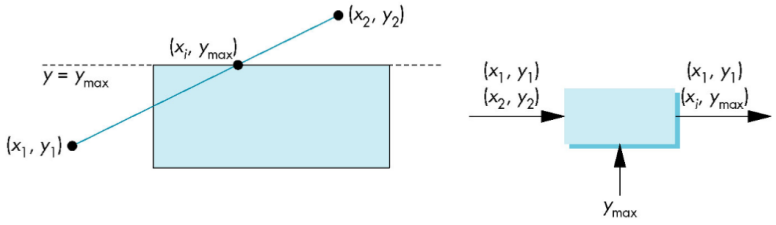
凸面与曲面细分（Convexity and tessellation）

一种方法是用一组三角形多边形替换原来的多边形。从而使得仅有凸多边形，容易填充。tessellation的代码在GLU Library中。



Clipping as a black box

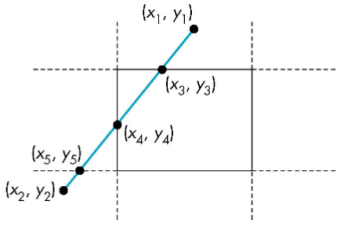
We can consider line segment clipping as a process that takes in two vertices and produces either no vertices or the vertices of a clipped line segment.

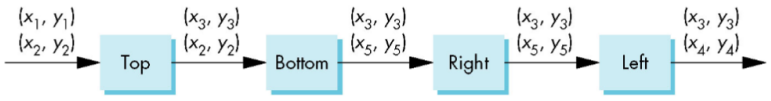


裁剪的渲染管线

线段:

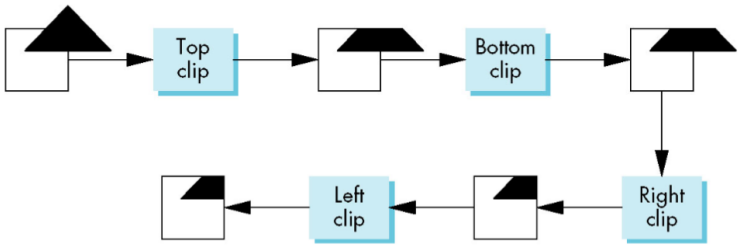
对窗口的每一侧进行裁剪是独立于其他侧的，因此我们可以在一个管道中使用四个独立的裁剪器。





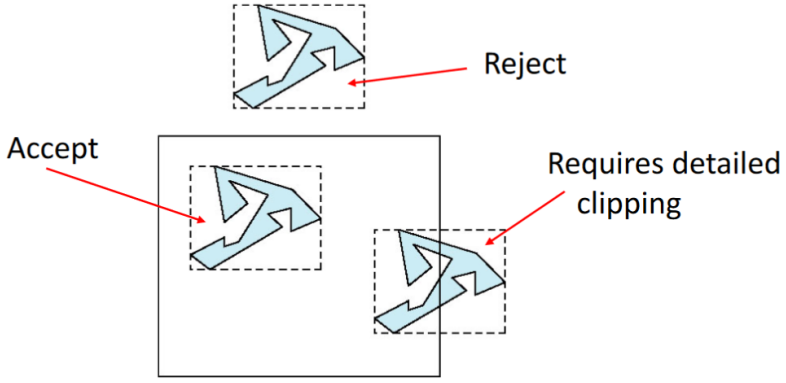
多边形:

若有3个维度则添加前面和后面的2个裁剪。下图是SGI几何引擎的策略，延迟略有增加:



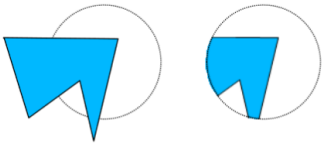
边界框（Bounding box）

可以使用轴对齐（axis-aligned）的边界框（容易计算x和y的最大最小值）来判断多边形是否需要进行详细的裁剪：

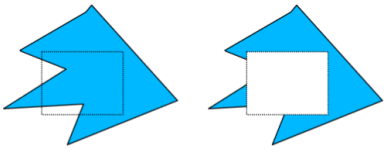


其他问题：

- Clipping other shapes: Circle, Ellipse, Curves
- Clipping a shape against another shape.



- Clipping the interior



OpenGL代码

在Opengl中设置2D窗口：

```
glViewport(xvmin, yvmin, vpWidth, vpHeight);
```

所有参数都以相对于显示窗口左下角的整数屏幕坐标给出。如果我们不调用这个函数，默认情况下，使用与显示窗口大小和位置相同的视口（即，所有 GLUT 窗口都用于 OpenGL 显示）。

在 OpenGL 中设置剪切窗口

```
glMatrixMode(GL_PROJECTION)
glLoadIdentity(); // reset, so that new viewing parameters
                  // are not combined with old ones (if any)
gluOrtho2D(xwmin, xwmax, ywmin, ywmax);
glOrtho(xwmin, xwmax, ywmin, ywmax, near, far);
gluPerspective(vof, aspectratio, near, far);
glFrustum(xwmin, xwmax, ywmin, ywmax, near, far);
```

剪切窗口中的对象将转换为归一化坐标 （-1,1）。

总结：

- 剪切的概念：什么是裁剪，重要性；点线与多边形
- 线段裁剪算法：暴力联立方程（Brute Force Simultaneous Equations），暴力相似三角形（Brute Force Similar Triangles），Cohen-Sutherland，Liang-Barsky
- 多边形裁剪（简述）
- Opengl函数：

  glViewport()
  glMatrixMode() / glLoadIdentity()
  glOrtho() / gluOrtho2D()
  gluPerspective / glFrustum(