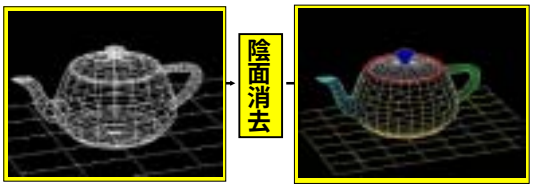
コンピュータグラフィックス特論
(ビジュアルコンピューティング論)
(エンターテインメントテクノロジー研究II)

Hidden surface removal

T. Nishita

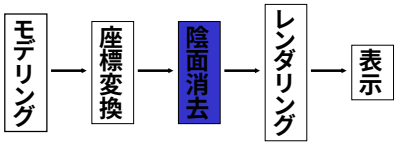
CG画像生成パイプライン

- 画像生成過程



CG画像生成パイプライン

- 画像生成過程

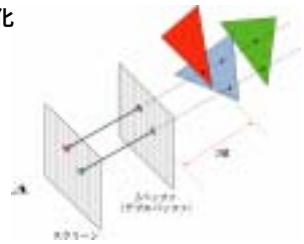


代表的方法

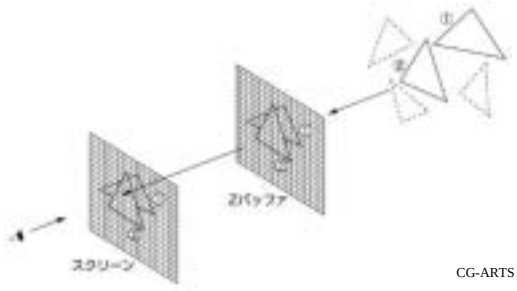
- Z-Buffer法
- Zソート法
- スキャンライン法
- レイトレーシング法

Zバッファ法

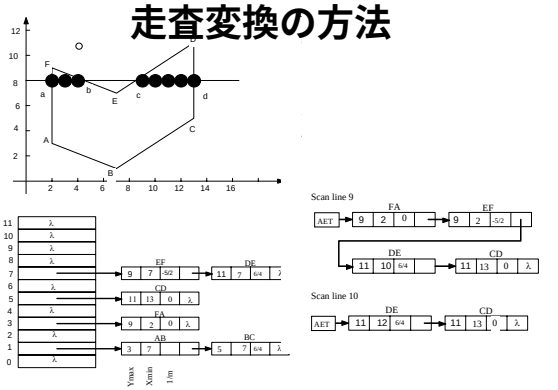
- Zバッファを無限遠で初期化
- 各ポリゴンについて
  - 透視変換
  - 各画素(i,j)について
    - Z値の計算
    - If($z < z_{buff}(i,j)$)
      - 描画
      - $Z_{buff}(i,j) = z$
- ほとんどのハードに搭載
- ポリゴンならどんな場合もOK



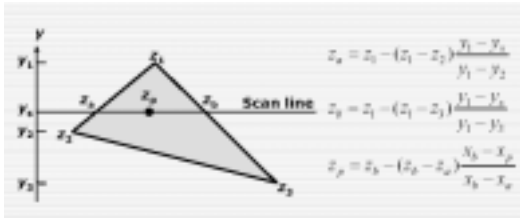
Depth Buffer法(Z-buffer法)



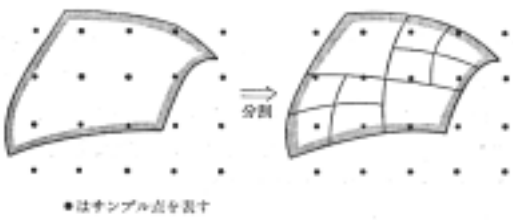
走査変換の方法



奥行きの計算

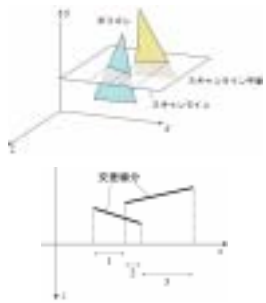


曲面のzバッファ法



スキャンライン法

- 全てのポリゴンを透視変換
- 各スキャンラインについて
  - スキャンライン平面とポリゴンの交線算出
  - 陰線消去して描画
- 一次元の問題に帰着
- ポリゴンならどんな場合もOK



曲面のスキャンライン法

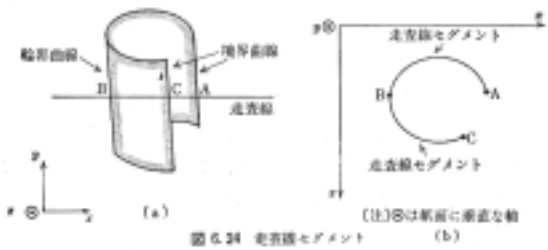
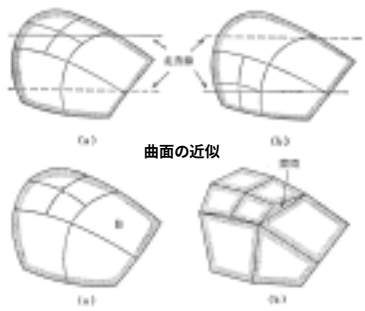


図 6.34 走査線セグメント

曲面の再分割近似法

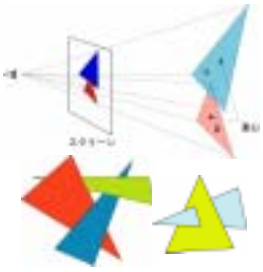


曲面の近似

曲面を独立に処理するために発生する隙間

Zソート法

- 全てのポリゴンについて（例えば）重心位置のZ値を計算
- Z値の大きい順にソート
- 各ポリゴンについて
  - 透視変換
  - 描画（各画素の内容を上書き）
- 単純だがうまくいかない場合がある



Zソート法；Depth Sort法 (painter's algorithm)

- 物体を視点からの距離にしたがってsortする
- 遠い物体から順に重ね書きをする

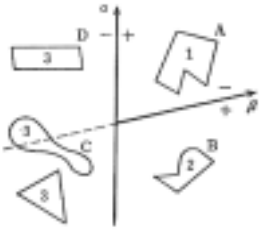
- 距離によるsortが可能であれば，きわめて高速
- 遠景，中景，近景がはっきりしているときに有効

優先順位アルゴリズム

問題が生じる場合

サイクリックな優先順位

優先順位アルゴリズム

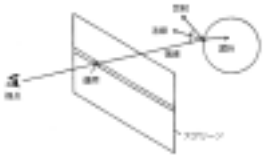


クラスタ間の優先順位の決定

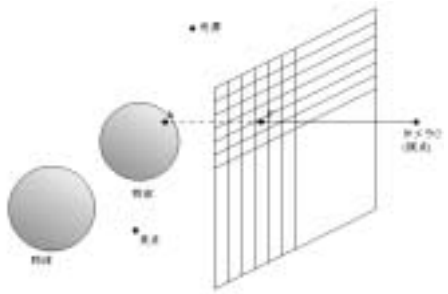
クラスタ間の優先順位の決定
BSP（**B**inary **S**pace-**P**artitioning）法と呼ばれる

レイトレーシング法

- 各画素について
  - 視線を算出
  - 視線と全物体の交点計算
  - 視点に最も近い交点を算出
  - 描画
- ポリゴンでなくともよい
- 反射／屈折を表現
- 一般には，とても遅い



レイトレーシング法（光線追跡法）

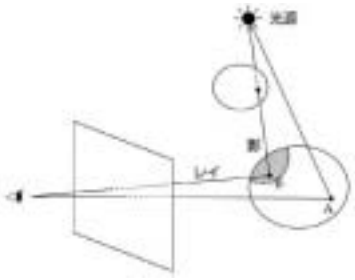


光線追跡法の特徴

- 計算量が多い → 高速化
- algorithmがsimple
- 不必要な光線は追跡しない
- 重要性の少ない光線の追跡が多い

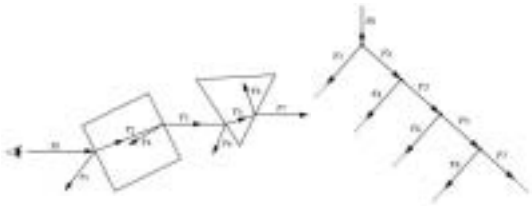
```
for すべてのpixel P do
  Pと視点Vを通る直線をLとする;
  for すべての物体O do
    LとOの交点を求める;
  if ひとつでも交点があった
    then 視点にもっとも近い交点をIとする;
         Iと光源Sとの間に他の物体がある
           then PにOの色＋環境光をぬる
           else Iにshadingの計算を行い，
                Pにその色をぬる
    else Pに背景色をぬる;
```

影の表示



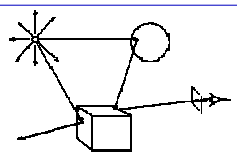
CG-ARTS

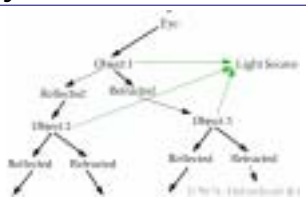
光線追跡法

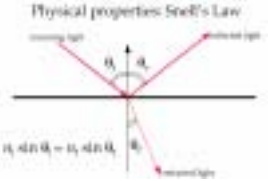


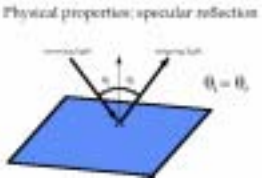
CG-ARTS

Ray Tree

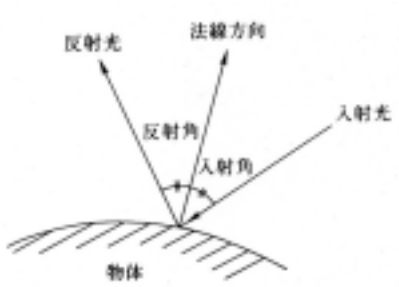




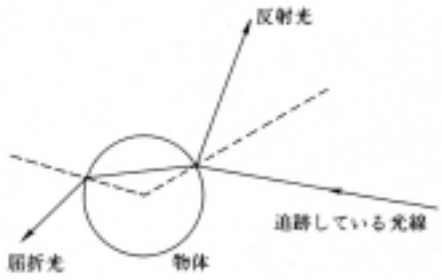


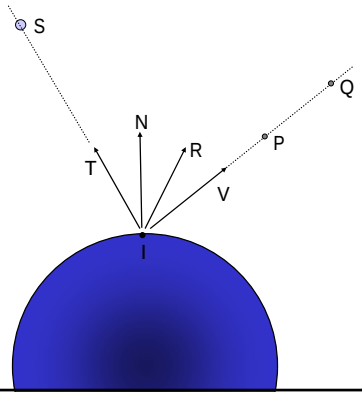


光の反射



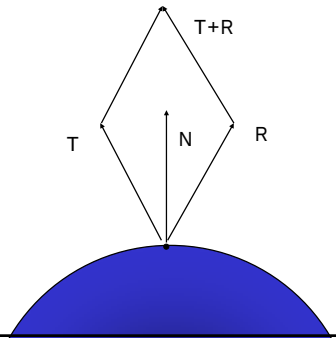
反射光･屈折光の追跡



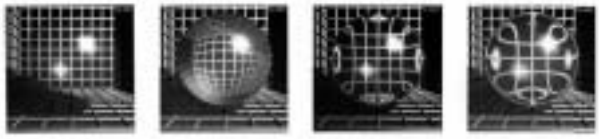


T+R=2N(N･T)

R=2N(N･T)-T



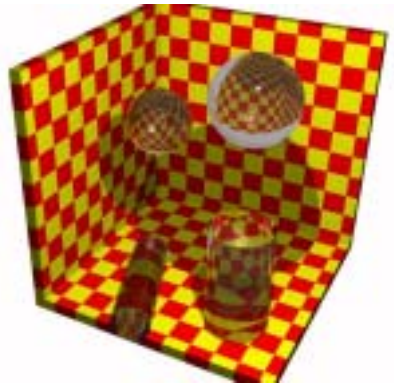
主な物体の屈折率



空気 1.00　**水** 1.33　**ガラス** 1.5〜1.7　**ダイヤ** 2.42

追跡打ち切りの基準

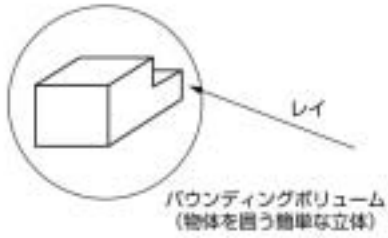
- 交点がなかった
- 拡散反射面にぶつかった
- 強さが一定のしきい値以下になった
- 反射回数が一定のしきい値以上になった



光線追跡法の高速化

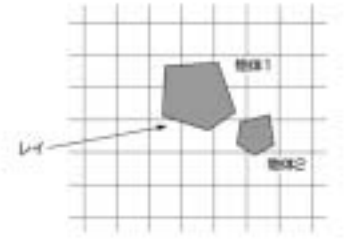
- hardwareによる方法
  - 並列computerの利用
- algorithmの工夫による方法
  - bounding volume法 → 交点計算1回あたりの平均時間削減
  - 空間分割法 → 交点計算の回数削減

Bounding Volume法



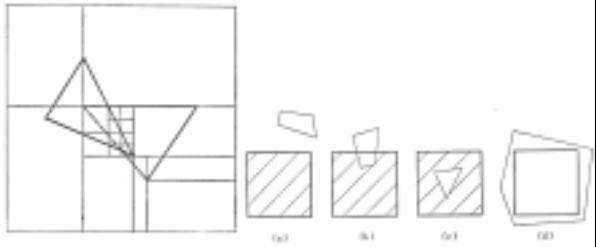
CG-ARTS

空間分割法



CG-ARTS

領域細分アルゴリズム

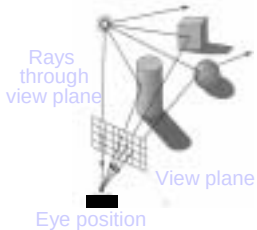


長方形領域への分割例　　ウィンドウと多角形の関係

Ray Casting

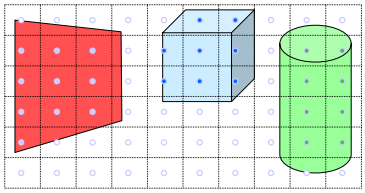
•A simple form of Ray Tracing

Simplest method is ray casting



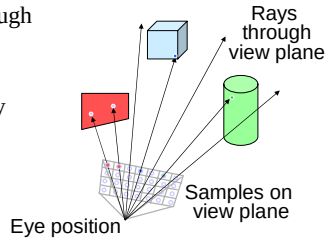
Ray Casting

- To create each sample …
  - Construct ray from eye position through view plane
  - Find first surface intersected by ray through pixel
  - Compute color sample based on surface radiance



Ray Casting

- For each sample …
  - Construct ray from eye position through view plane
  - Find first surface intersected by ray through pixel
  - Compute color sample based on surface radiance

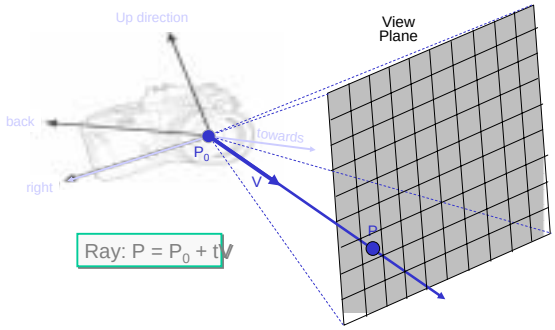


Ray Casting

- Simple implementation:

```
Image RayCast(Camera camera, Scene scene, int width, int height)
{
    Image image = new Image(width, height);
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            Ray ray = ConstructRayThroughPixel(camera, i, j);
            Intersection hit = FindIntersection(ray, scene);
            image[i][j] = GetColor(hit);
        }
    }
    return image;
}
```

Constructing Ray Through a Pixel

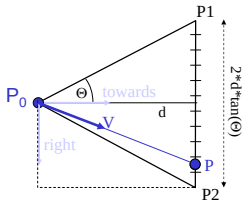


Constructing Ray Through a Pixel

- 2D Example

Θ = frustum half-angle
d = distance to view plane

right = towards x up



$P1 = P_0 + d*towards - d*tan(\Theta)*right$
$P2 = P_0 + d*towards + d*tan(\Theta)*right$

$P = P1 + (i/width + 0.5) * (P2 - P1)$
$= P1 + (i/width + 0.5) * 2*d*tan(\Theta)*right$
$V = (P - P_0) / \|P - P_0\|$

Ray: $P = P_0 + tV$

Ray Casting

- Simple implementation:

```
Image RayCast(Camera camera, Scene scene, int width, int height)
{
    Image image = new Image(width, height);
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            Ray ray = ConstructRayThroughPixel(camera, i, j);
            Intersection hit = FindIntersection(ray, scene);
            image[i][j] = GetColor(hit);
        }
    }
    return image;
}
```

Ray-Scene Intersection

- Intersections with geometric primitives
  - Sphere
  - Triangle
  - Groups of primitives (scene)

光線と球との交点

- **球の表現**: $(x-x_C)^2+(y-y_C)^2+(z-z_C)^2-r^2=0$
  - **球の中心**: $C=(x_C,y_C,z_C)$
  - **球の半径**: r
- **直線の表現**: $L(t)=(1-t)Q+tP=(P-Q)t+Q$

$$x_L(t)=(x_P-x_Q)t+x_Q,$$
$$y_L(t)=(y_P-y_Q)t+y_Q,$$
$$z_L(t)=(z_P-z_Q)t+z_Q$$

  - **ピクセル**: $P=(x_P,y_P,z_P)$
  - **視点**: $Q=(x_Q,y_Q,z_Q)$

球面の式に直線の式を代入

$$\begin{aligned}
&(x-x_C)^2+(y-y_C)^2+(z-z_C)^2-r^2\\
&=\{(x_P-x_Q)t+x_Q-x_C\}^2+\{(y_P-y_Q)t+y_Q-y_C\}^2\\
&\quad+\{(z_P-z_Q)t+z_Q-z_C\}^2-r^2\\
&=\{(x_P-x_Q)^2+(y_P-y_Q)^2+(z_P-z_Q)^2\}t^2\\
&\quad+2\{(x_P-x_Q)(x_Q-x_C)+(y_P-y_Q)(y_Q-y_C)+(z_P-z_Q)(z_Q-z_C)\}t\\
&\quad+\{(x_Q-x_C)^2+(y_Q-y_C)^2+(z_Q-z_C)^2-r^2\}\equiv 0
\end{aligned}$$

2次方程式の係数

$$\alpha t^2+\beta t+\gamma=0$$

$$\begin{aligned}
\alpha&=(x_P-x_Q)^2+(y_P-y_Q)^2+(z_P-z_Q)^2,\\
\beta&=2\{(x_P-x_Q)(x_Q-x_C)+(y_P-y_Q)(y_Q-y_C)\\
&\qquad+(z_P-z_Q)(z_Q-z_C)\},\\
\gamma&=(x_Q-x_C)^2+(y_Q-y_C)^2+(z_Q-z_C)^2-r^2,\\
D&=\beta^2-4\alpha\gamma \quad \text{あるいは} \quad D'=(\beta/2)^2-\alpha\gamma
\end{aligned}$$

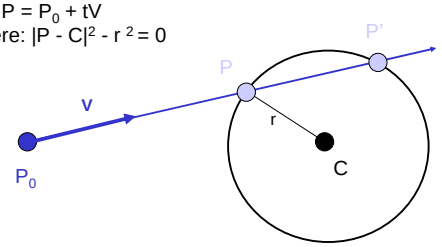
交点の算出

- $D<0$ のときは交点なし．$D\geqq 0$ のときは交わる．
- 交わるとき，2次方程式の解を t_1, t_2 とすると，$L(t_1)$, $L(t_2)$ が交点．
- 視点に近い交点は小さい方の解に対応する．

$$t=\frac{-\beta-\sqrt{\beta^2-4\alpha\gamma}}{2\alpha}$$

別解； 光線と球との交点
Ray-Sphere Intersection(1)

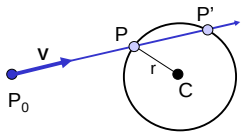
Ray: $P = P_0 + tV$
Sphere: $|P - C|^2 - r^2 = 0$



Ray-Sphere Intersection (2)

Ray: $P = P_0 + tV$
Sphere: $|P - C|^2 - r^2 = 0$

Substituting for P, we get:
$|\mathbf{P_0 + tV} - C|^2 - r^2 = 0$



Solve quadratic equation:
$at^2 + bt + c = 0$
where:
$a = |V|^2 = 1$
$b = 2\ V \bullet (P_0 - C)$
$c = |P_0 - C|^2 - r^2$

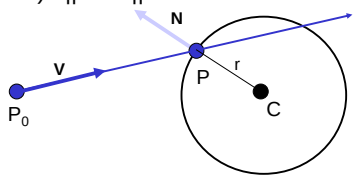
If ray direction is normalized!

$P = P_0 + tV$

Ray-Sphere Intersection (3)

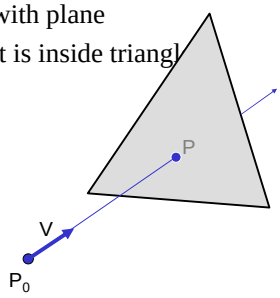
- Need normal vector at intersection for lighting calculations

$N = (P - C) / ||P - C||$



Ray-Triangle Intersection

- First, intersect ray with plane
- Then, check if point is inside triangl



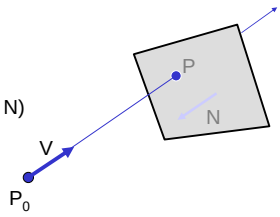
Ray-Plane Intersection

Ray: $P = P_0 + tV$
Plane: $P \bullet N + d = 0$

Substituting for P, we get:
$(\mathbf{P_0 + tV}) \bullet N + d = 0$

Solution:
$t = -(P_0 \bullet N + d) / (V \bullet N)$

$P = P_0 + tV$

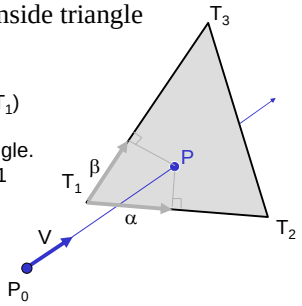


Ray-Triangle Intersectio

- Check if point is inside triangle parametrically

Compute α, β:
$P = \alpha\ (T_2 - T_1) + \beta\ (T_3 - T_1)$

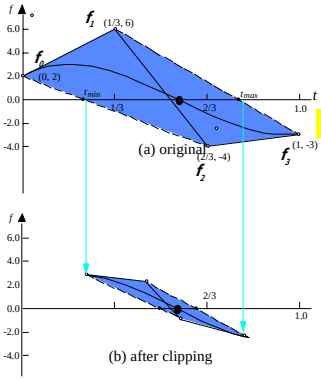
Check if point inside triangle.
$0 \le \alpha \le 1$ and $0 \le \beta \le 1$
$\alpha + \beta \le 1$



Other Ray-Primitive Intersections

- Cone, cylinder, ellipsoid:
  - Similar to sphere
- Box
  - Intersect 3 front-facing planes, return closest
- Convex polygon
  - Same as triangle (check point-in-polygon algebraically)
- Concave polygon
  - Same plane intersection
  - More complex point-in-polygon test

Bezier Clipping



$f(x)=24x^3-42x^2+12x+2=0$

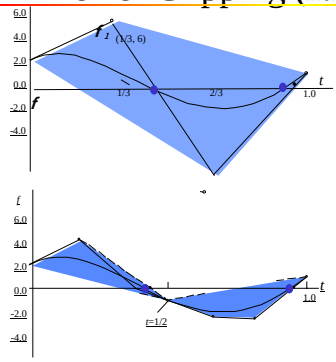
$$f(t)=\sum_{k=0}^{3} f_k B_k^3(t)$$

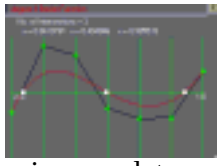
($i/n, f_k$):control point

t_{max} t_{min}

1st; 0.5555
2nd; 0.0083
3rd; 0.0003

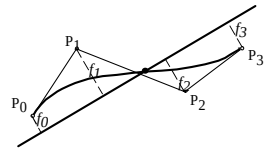
Bezier Clipping (multiple roots)





java applet

Curve-Line intersection



curve

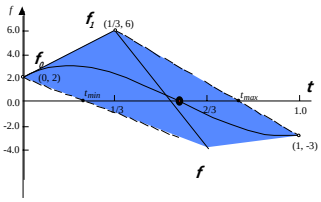
$$x(t)=\sum_{k=0}^{3} x_k B_k^3(t)$$
$$y(t)=\sum_{k=0}^{3} y_k B_k^3(t)$$

line

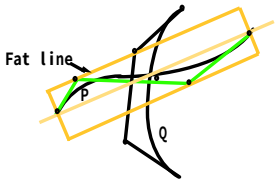
$d(x,y)=ax+by+c$

$$f(t)=\sum_{k=0}^{3} f_k B_k^3(t)$$

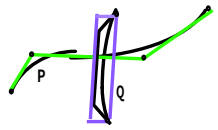
$f_k=ax_k+by_k+c$



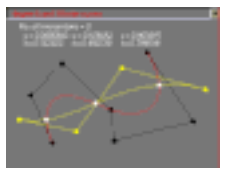
Curve/curve Intersection



(a) clip Q by Fatline of P



(b) clip P by Fatline of Q



Applications to 3D Rendering

rendering curved surfaces

parametric surfaces

x=x(u,v), y=y(u,v), z=z(u,v)

Bezier Patch, B-spline, NURBS

- **hidden line removal**
- **hidden surface removal**
  - **raytracing**
  - **scan line algorithm**

implicit surfaces

f(x,y,z) = 0

raytracing

algebraic surface
metaball (blobs)

Previous work for parametric patches

Raytracing

Whitted(1980): bicubic patches

subdivision approach

Kajiya(1982): bicubic patches

numerical solution

Nishita(1990): trimmed rational Bezier patches

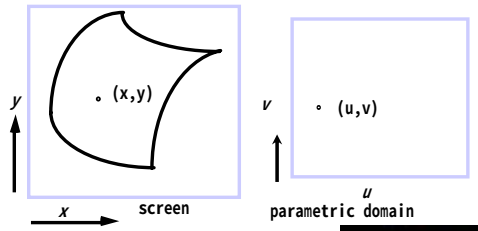
Bezier clipping

Scanline algorithm

Lane(1980): subdivision of curved surface into polygons at every scan line

Nishita(1991): modification of Lane's method (subdivision into subpatches at every scan line)

Raytracing for Trimmed Patchs

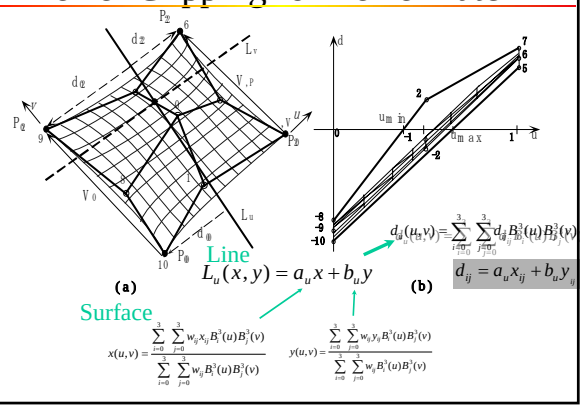


ray/patch intersection for bicubic patch;

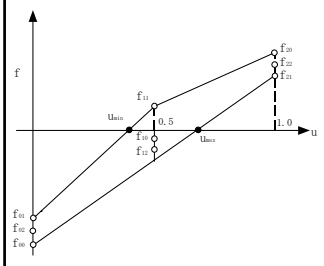
degree 18 of polynominal should be solved

trimmed patch

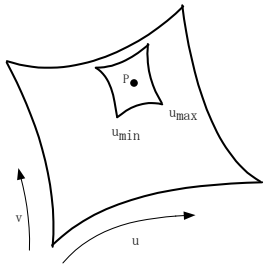
Bezier Clipping for Bezier Patch



Bezier Clipping for Bezier Patch

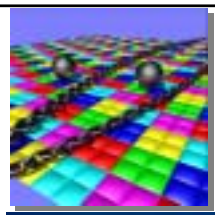


Extract intervals



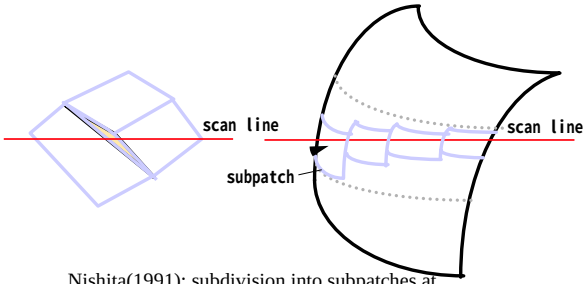
Iterative clipping

Raytracing using Bezier Clipping

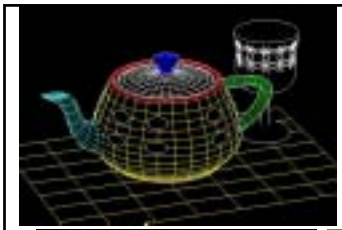


Scan Line Algorithm for Bezier Patches

Lane(1980): subdivision of curved surface into polygons at every scan line



Nishita(1991): subdivision into subpatches at every scan line



Scanline algorithm using Bezier Clipping

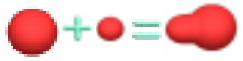



Metaball

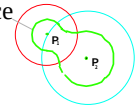
Modeling of curved surfaces

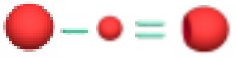
blobs; Blinn(1980)

metaballs; Nishimura(1985)



isosurface



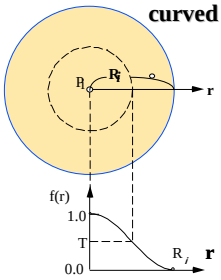


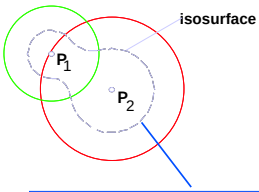
degree six field function

problem: ray/isosurface intersection

Metaball

curved surfaces defined by metaballs





$$f(x,y,z)=\sum_{i=0}^{n} q_i f_i - T = 0$$

T :threshold q :density

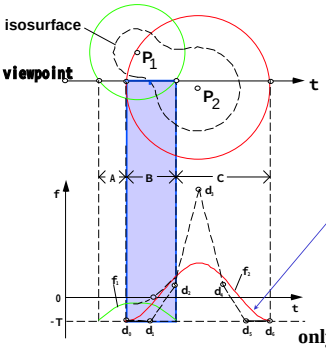
degree six field function;

degree six field function;

$f(0)=1,\ f'(0)=0,\ f(R_i)=0,\ f'(R_i)=0,\ f(R_i/2)=1/2$

q :density T :threshold

Intersection Test between Ray and Metaball



$$f_i(s_i)=\sum_{k=0}^{6} d_k^i B_k^6(s_i)$$

$d_0=d_1=d_5=d_6=0$

$d_2=d_4=\frac{16}{27}a_i^2$

$d_3=\frac{(8a_i+5)a_i^2}{45}$

only one intersection point closest to viewpoint is required

計算時間の比較

Algorithm	Number of Polygons		
	100	2,500	60,000
Depth sort	1	10	507
z-buffer	54	54	54
Scan line	5	21	100
Warnock area subdivision	11	64	307

Rendering法の比較

	Z-buffer 法	scan line 法	光線追跡法
速　度	やや速	速	遅
memory 量	多	少	少
手　順	単純	複雑	単純
反射・屈折	困難	困難	容易
対象物体	何でも	多面体が主	何でも