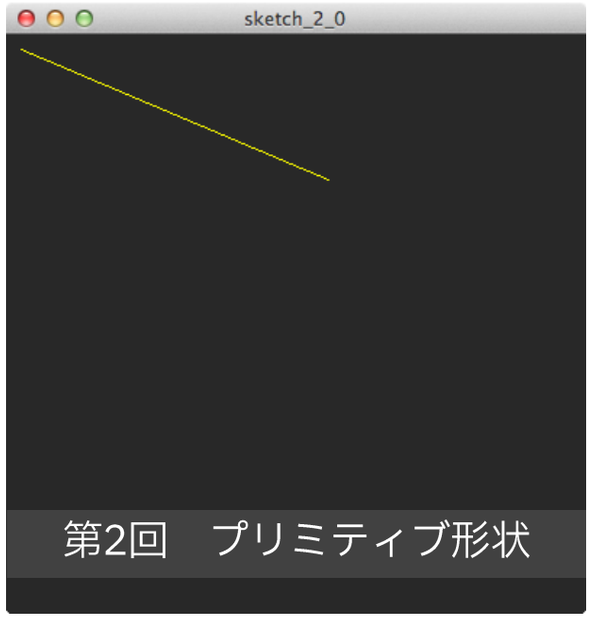
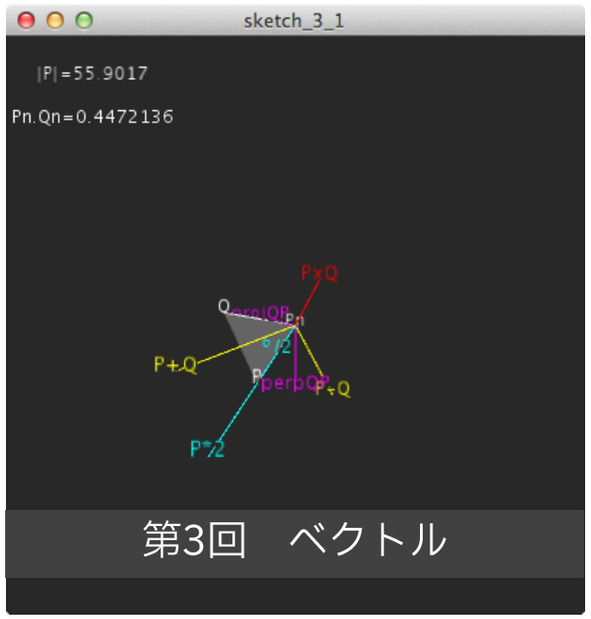
CGとCADの数理

GEOMETRIC MODELING AND COMPUTER GRAPHICS

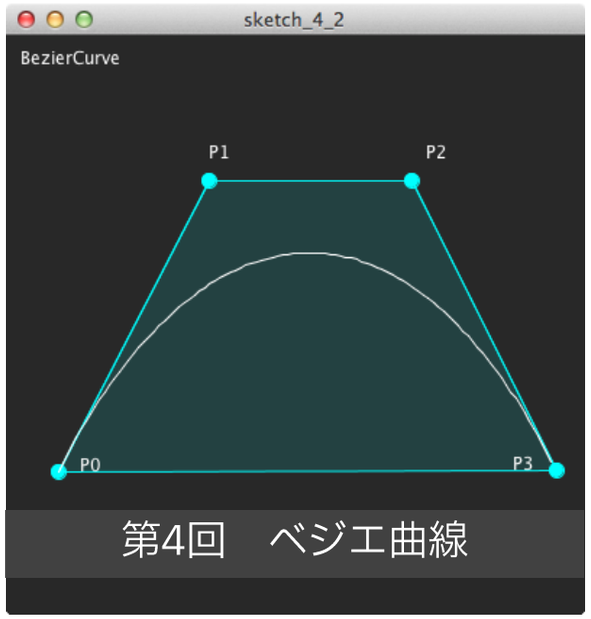
第12回 Zバッファ法



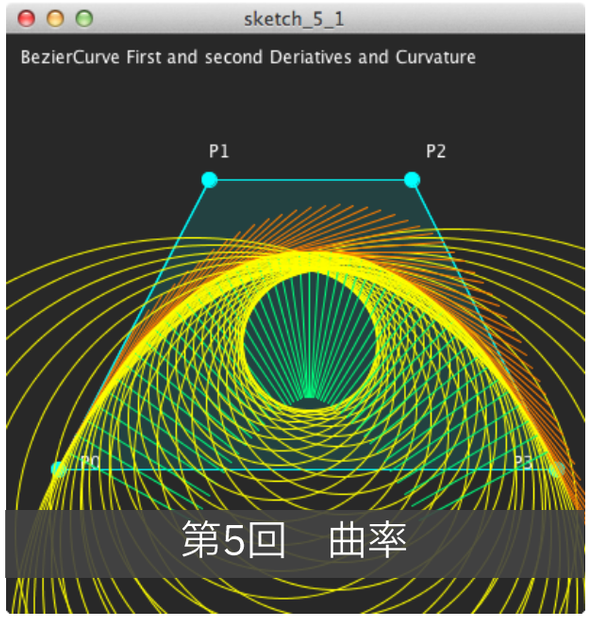
第2回　プリミティブ形状



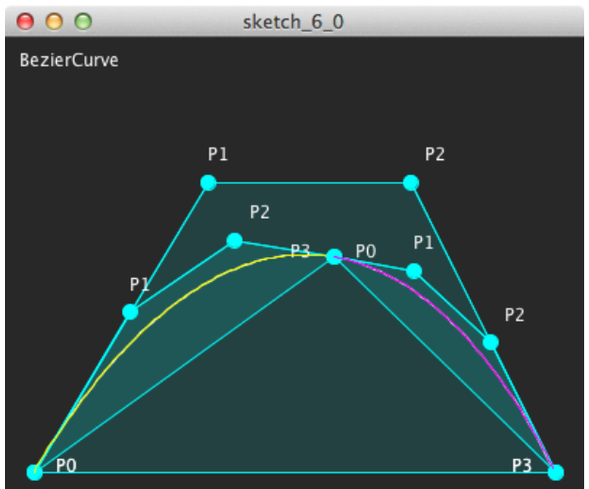
第3回　ベクトル



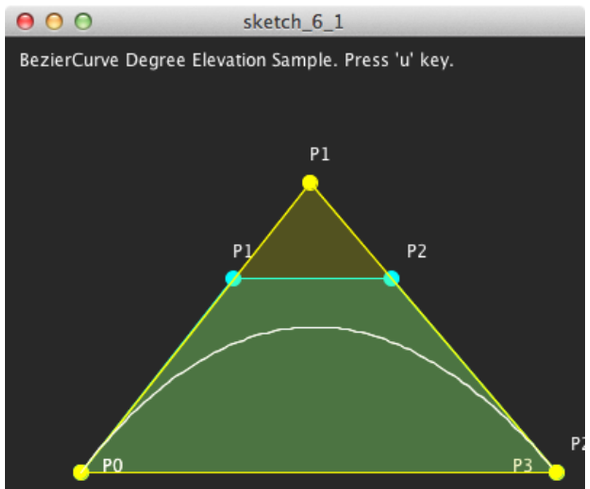
第4回　ベジエ曲線

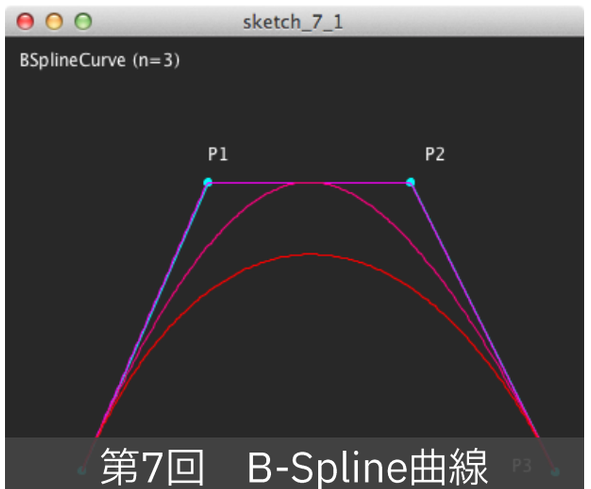


第5回　曲率



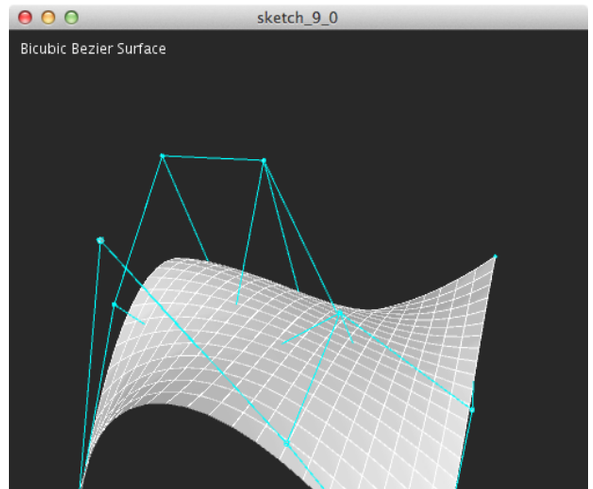
第6回　次数上げ、分割と接続





第7回　B-Spline曲線

第8回　NURBS曲線

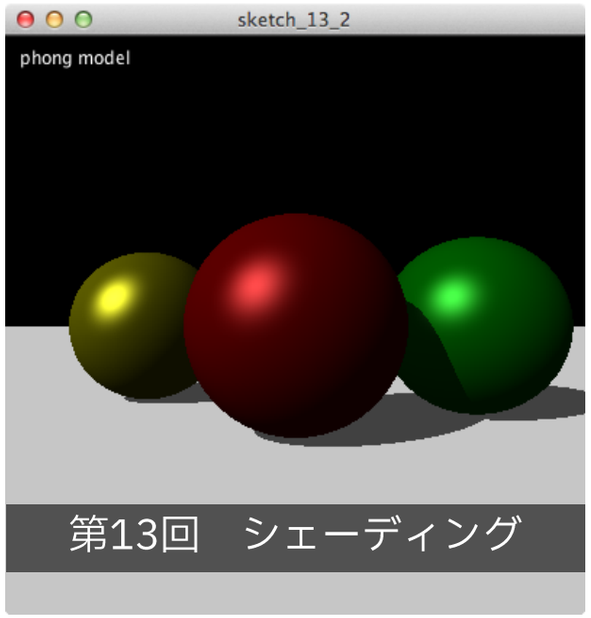


第9回　パラメトリックな曲面

第10回　行列

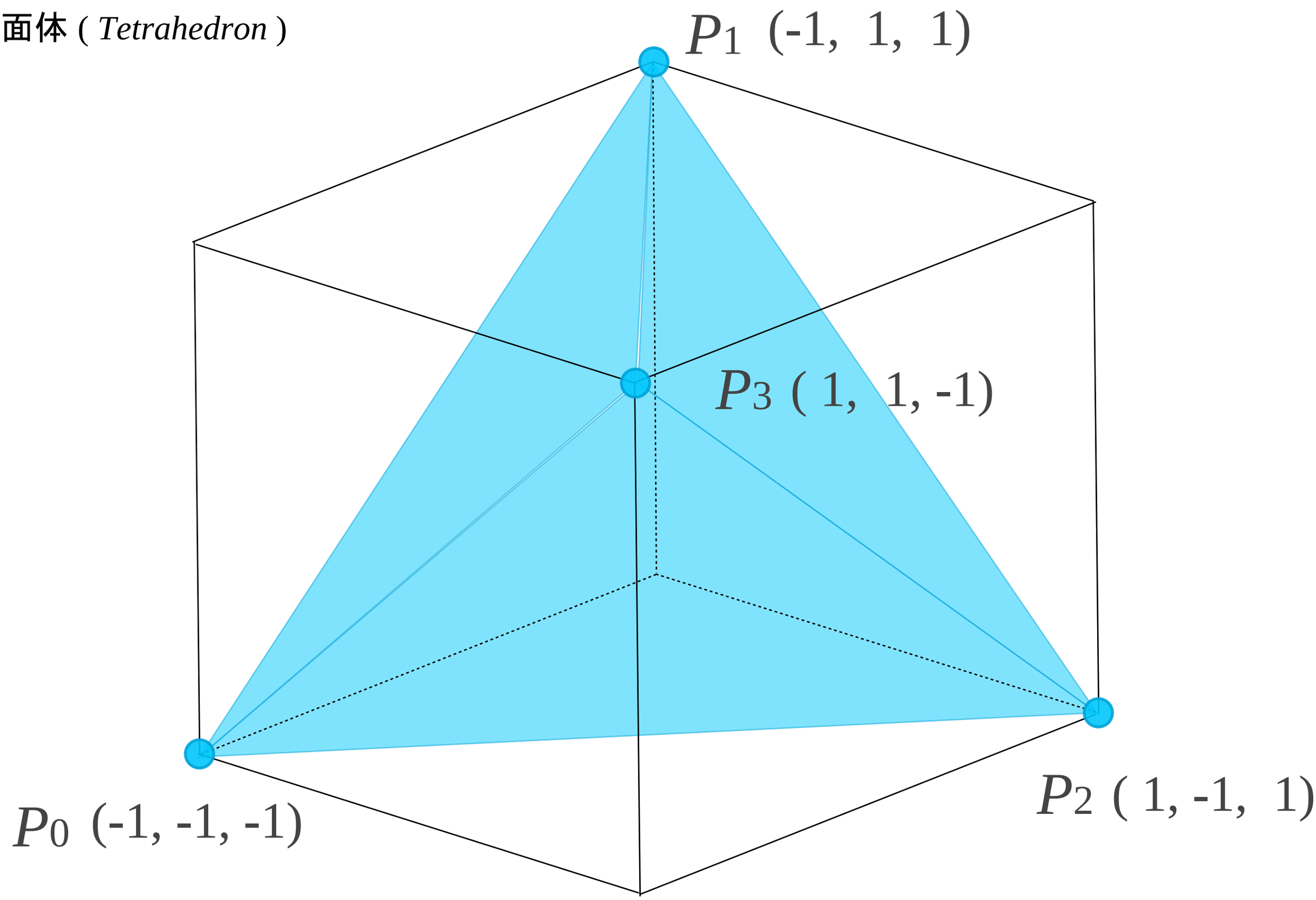
第11回　三次元物体の表示

第12回　レイトレーシング



第13回　シェーディング

四面体 (*Tetrahedron*)

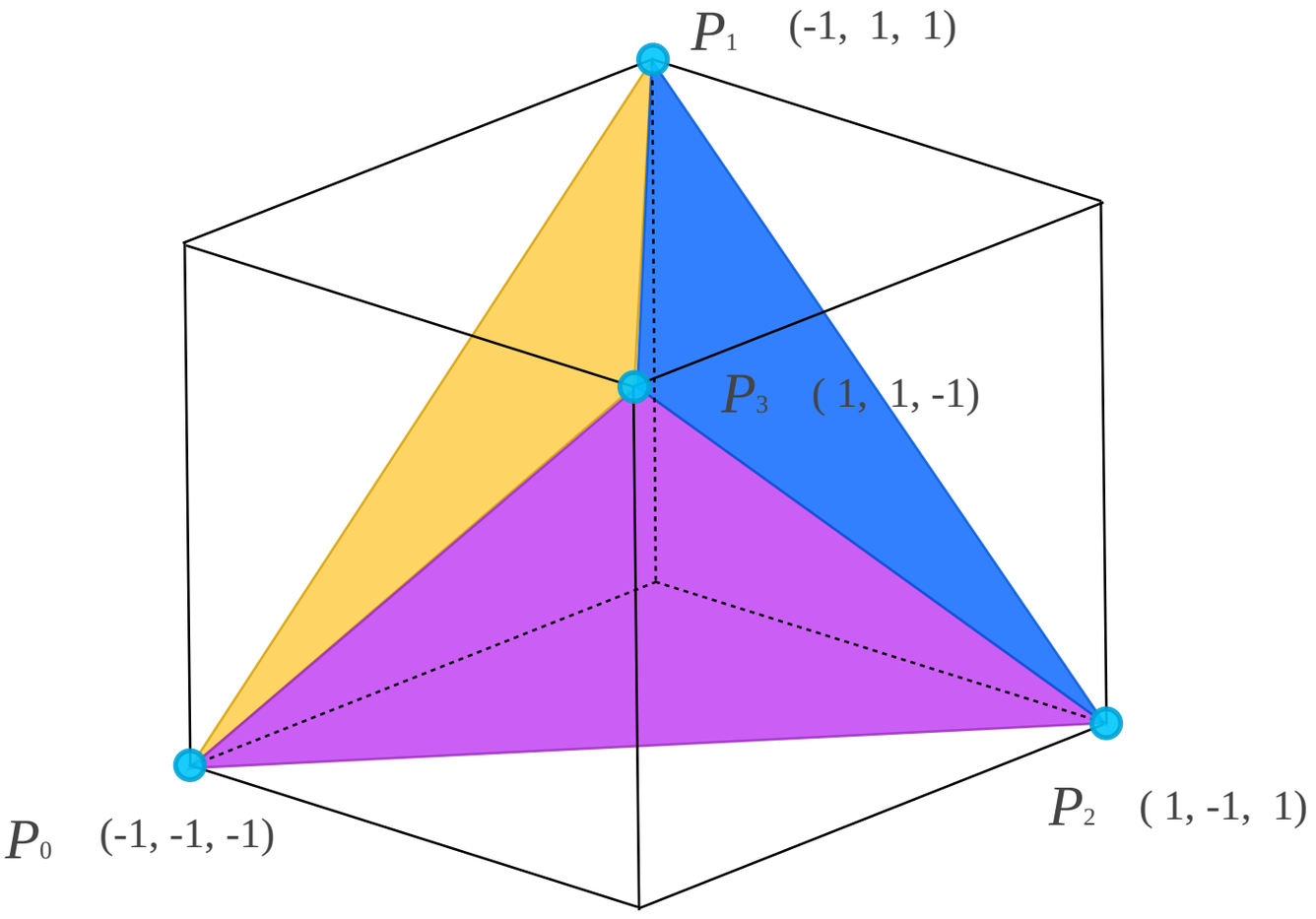


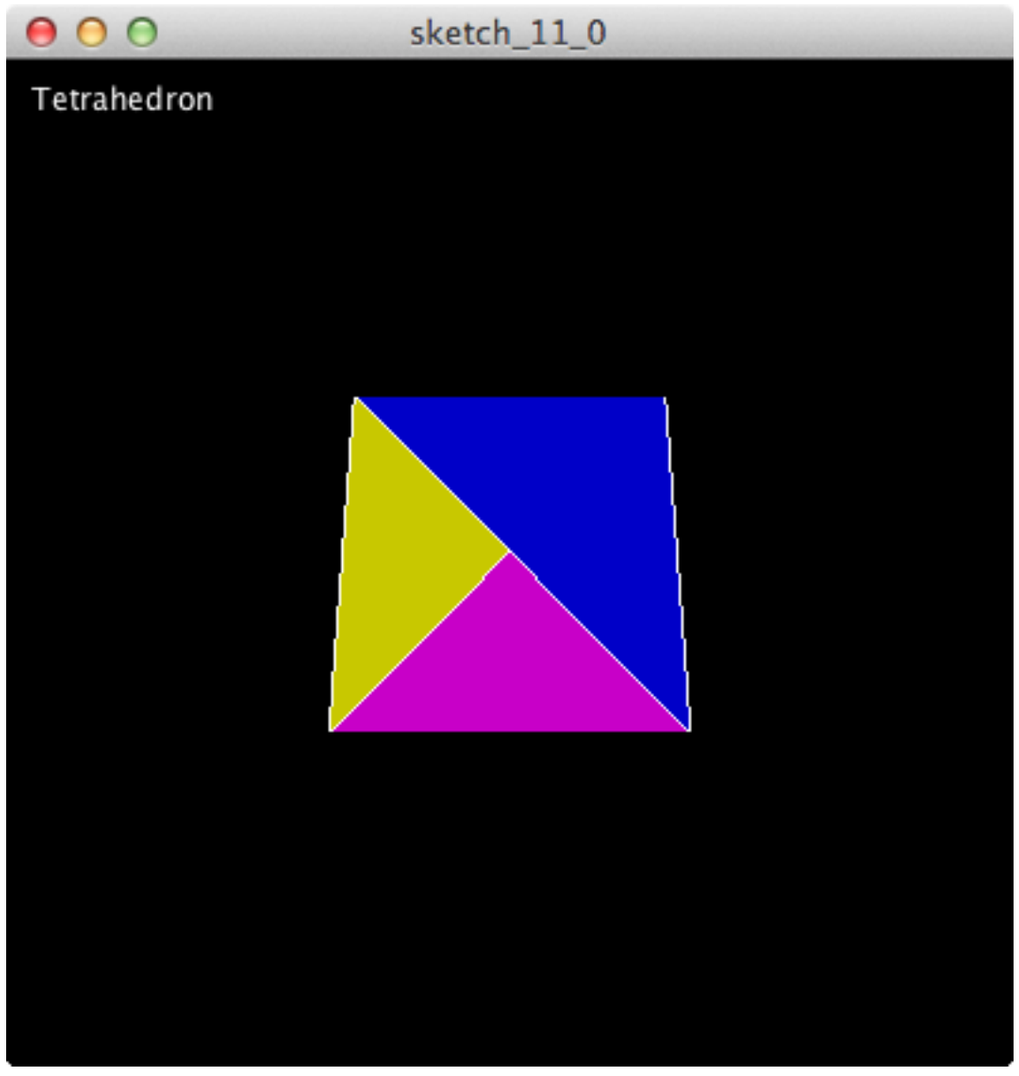
正多面体（*Regular Polyhedron*）とは、全ての面が一種類の多角形で出来てる凸多面体のことです。この正多面体は５種類しかありません。講義では正四面体を例に進めます。

sketch_12_0.zip をダウンロードして下さい

四面体 (*Tetrahedron*) を回転してみよう

```
init_matrix(M);                  // 行列の初期化（単位行列にします）
matrix_rotate(M, 'Y', r);        // Y軸中心の回転行列を作ります
Tt0.draw();                      // 四面体を描画します
```





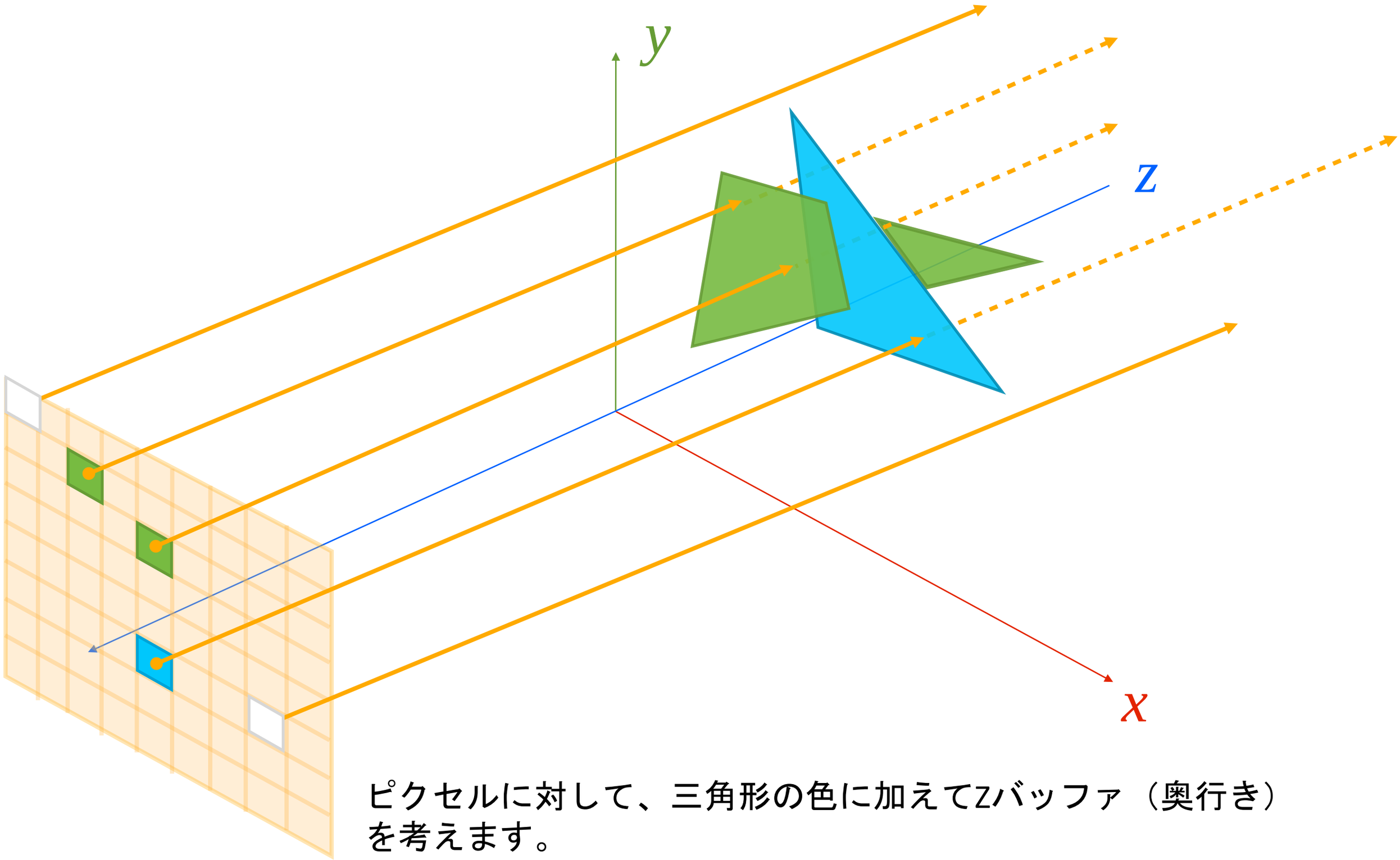
手前に見える物体だけを描画し、見えない物体は描画しない処理が必要です。

= 隠面処理（***Hidden-Surface Removal***）

古典的手法として、Zバッファアルゴリズム、スキャンライン法などが有名です。

sketch_11_1.zip をダウンロードして下さい

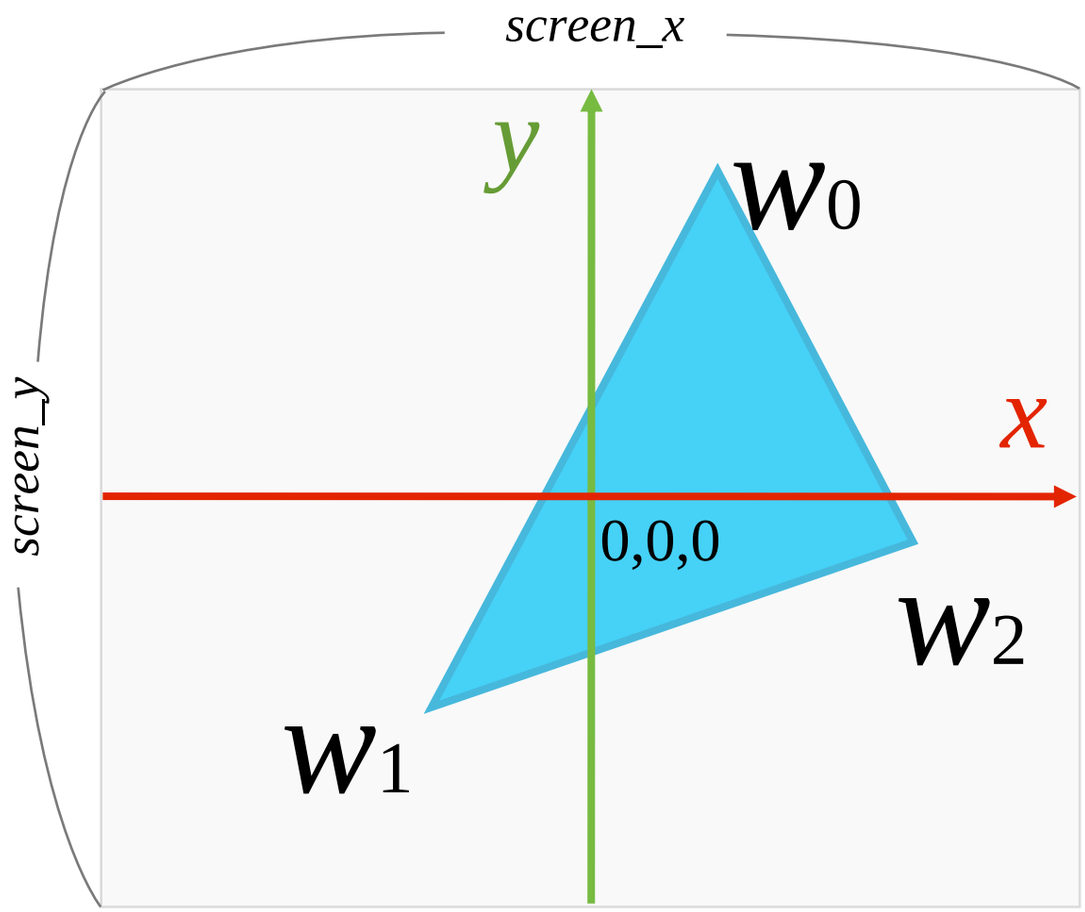
Zバッファ法のイメージ



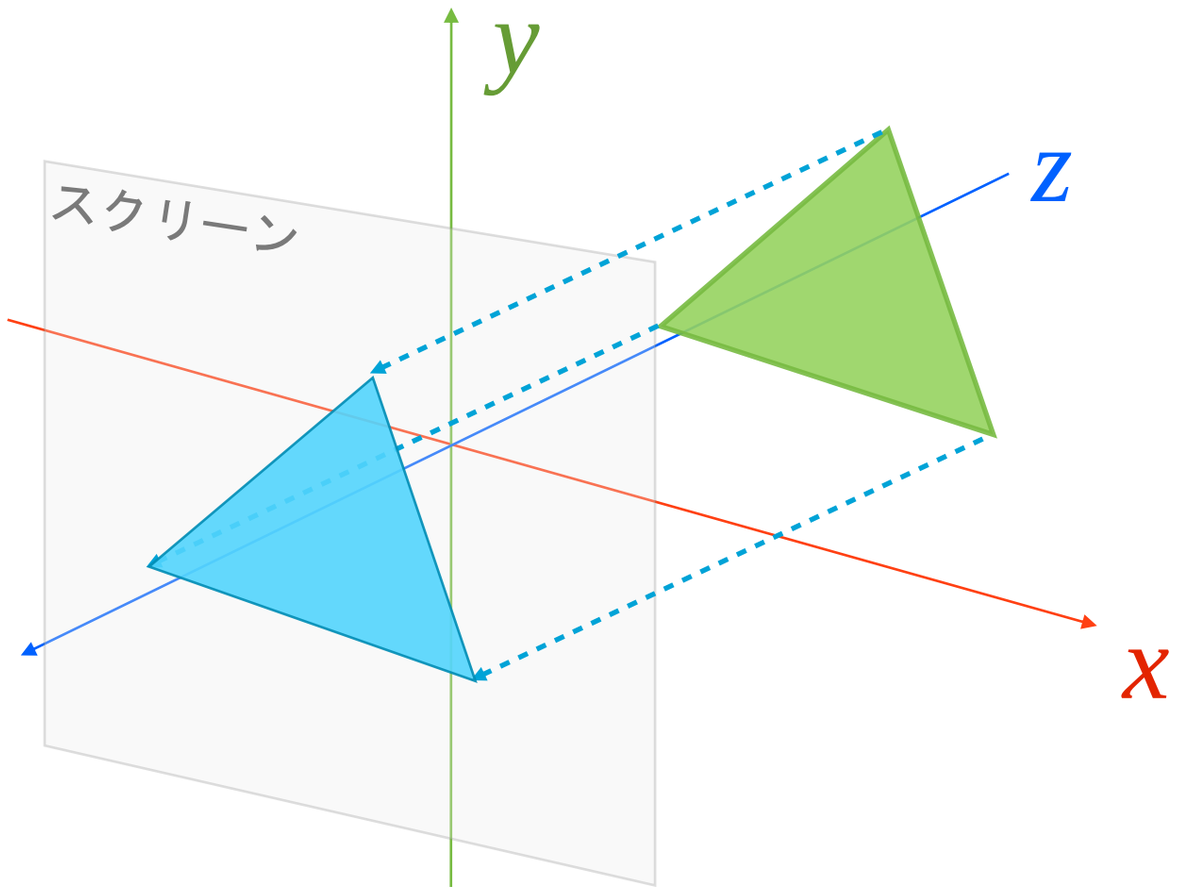
ピクセルに対して、三角形の色に加えてZバッファ（奥行き）を考えます。

```
float[][] z_buffer; //Zバッファ
z_buffer[j][i]= Float.NEGATIVE_INFINITY;
```

スクリーン座標系



ワールド座標系



スクリーン座標系への投影方法は、２種類あります。

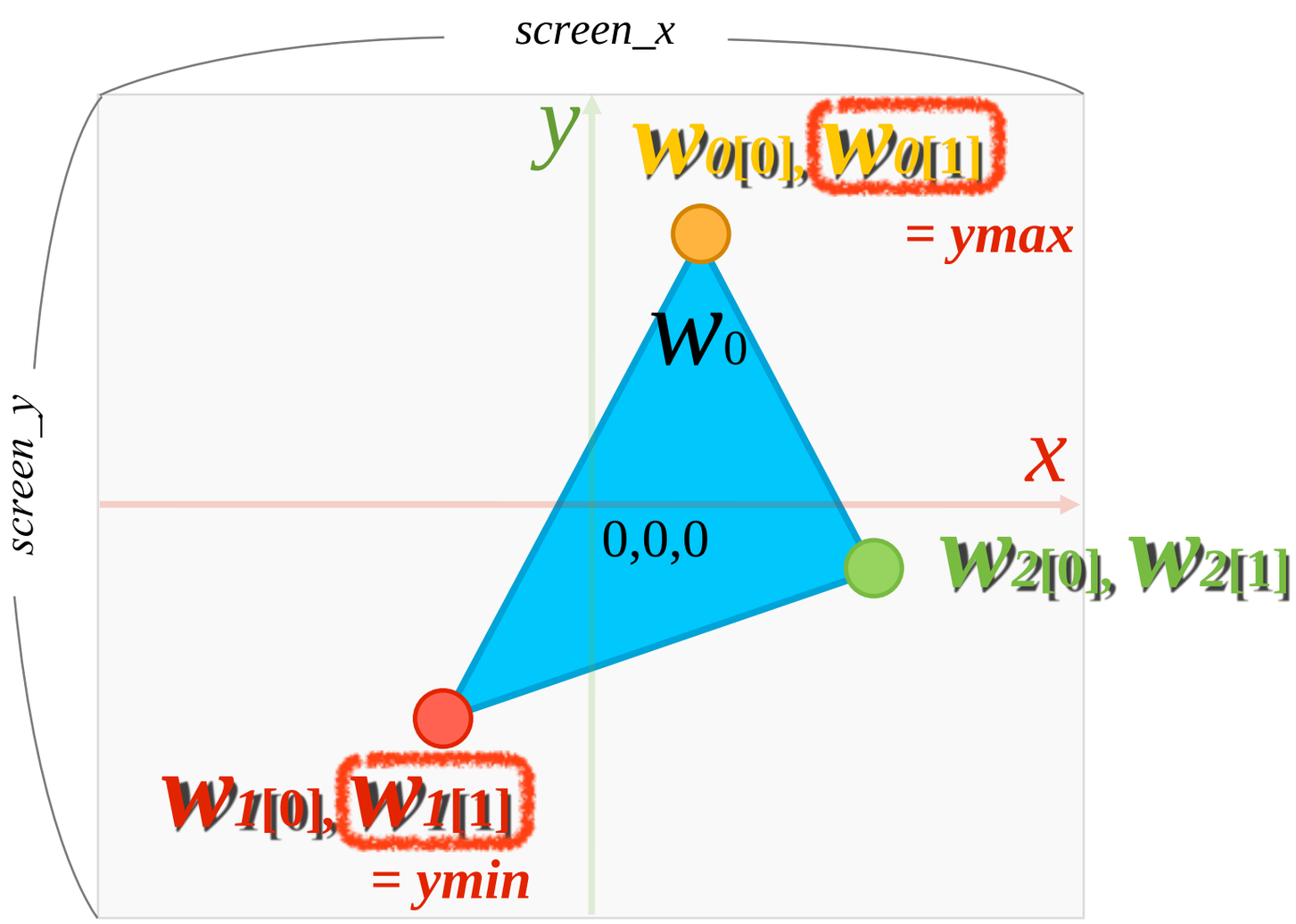
1. 平行投影（*Orthogonal Projection*）

    視点が無限遠にある場合の映像がスクリーンに映ると考えます。物体は遠くに合っても近くにあっても同じ大きさに見えます。

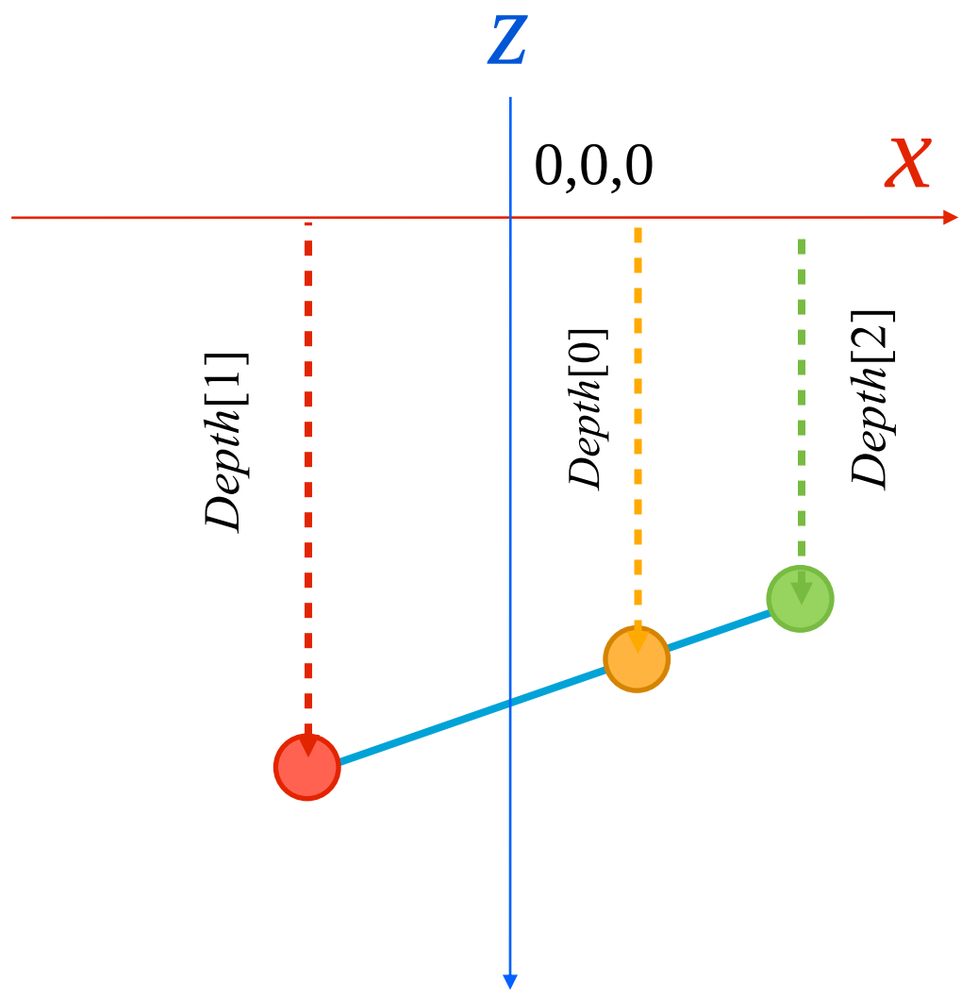
2. 透視投影（*Perspective Projection*）

    皆さんが使う”カメラ”で見たときに見える映像がスクリーンに映ると考えます。物体は遠くに行くほど小さく、近くに行くほど大きく見えます。

スクリーン座標系



ワールド座標系



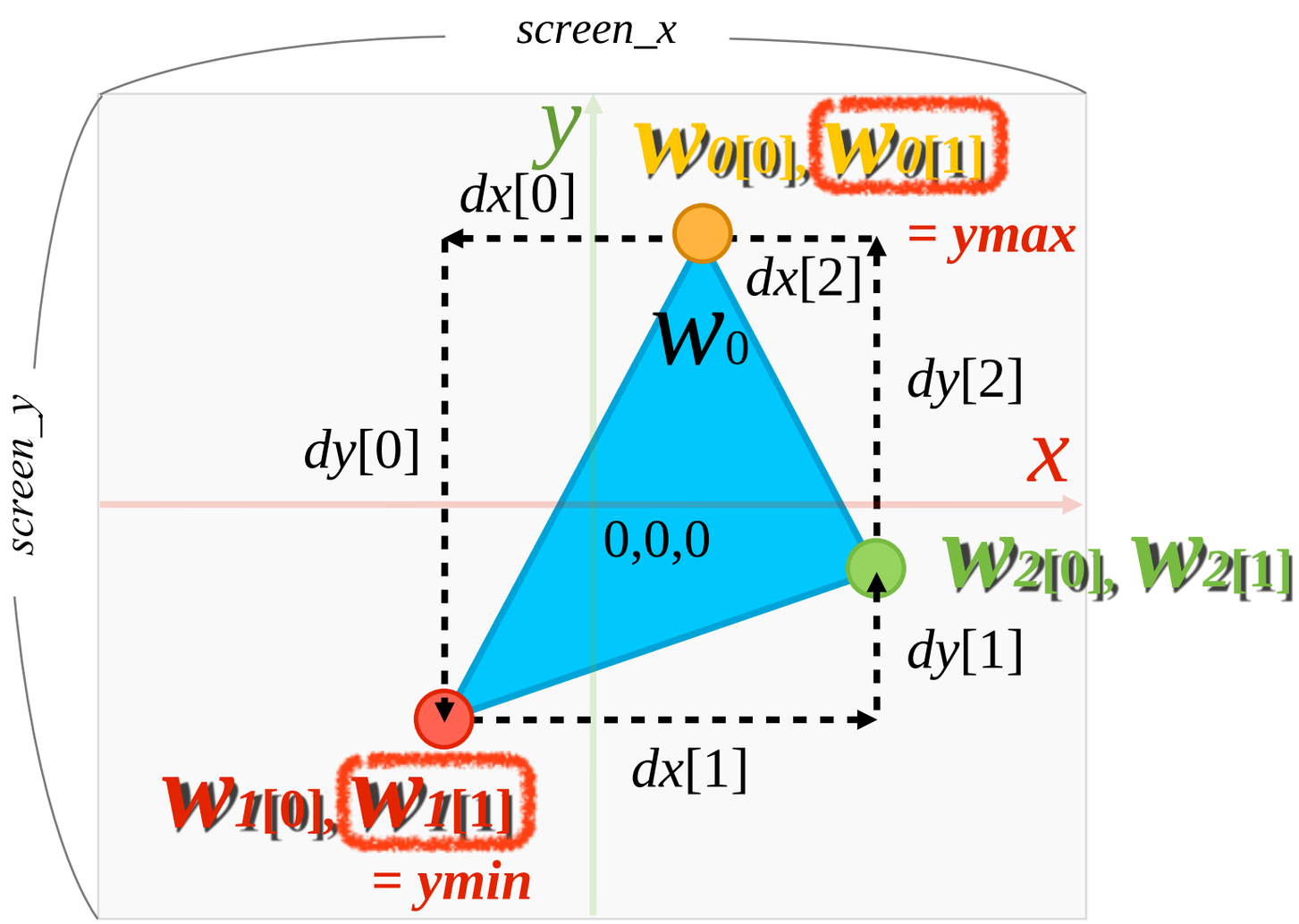

```
int ymin = (int)W0[1];
if(ymin>W0[1]){ymin=(int)W0[1];}
if(ymin>W1[1]){ymin=(int)W1[1];}
if(ymin>W2[1]){ymin=(int)W2[1];}
```

// Y座標の最小値を取得

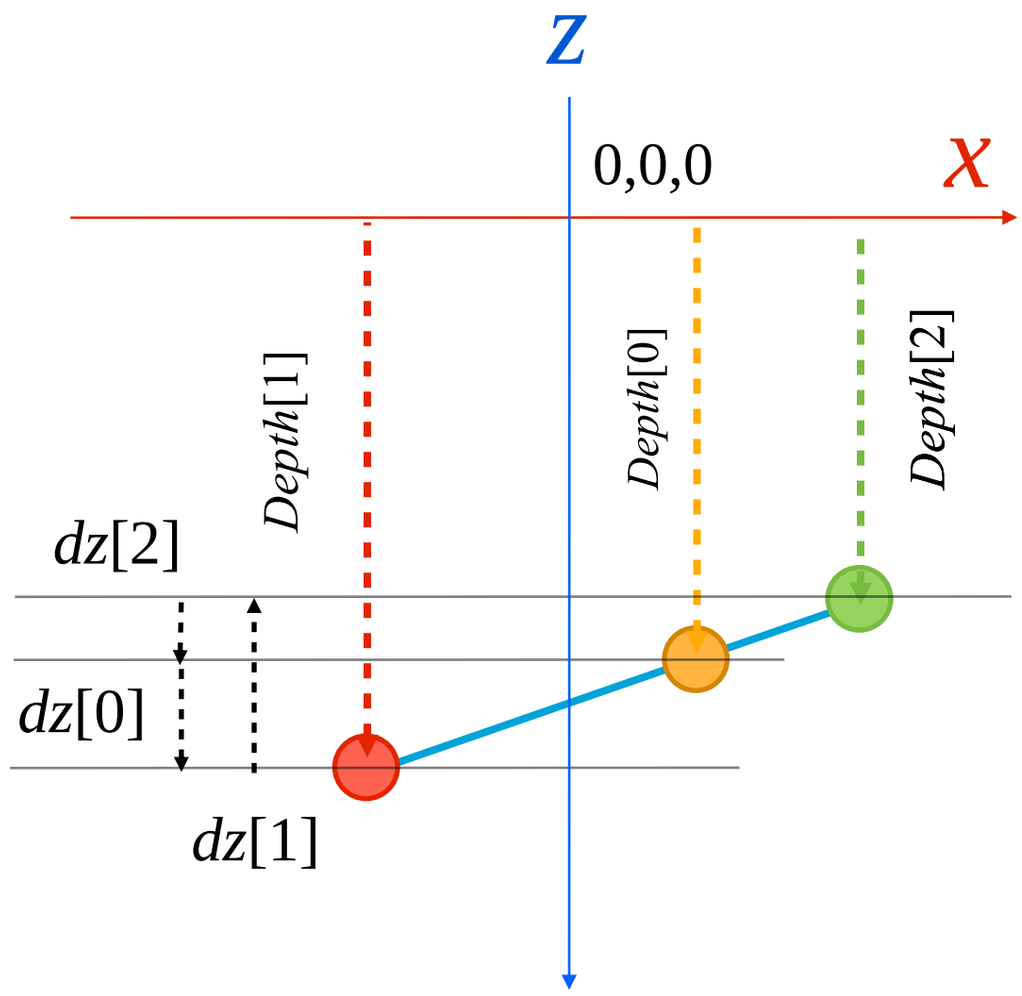
```
int ymax = (int)W0[1];
if(ymax<W0[1]){ymax=(int)W0[1];}
if(ymax<W1[1]){ymax=(int)W1[1];}
if(ymax<W2[1]){ymax=(int)W2[1];}
```

// Y座標の最大値を取得

スクリーン座標系



ワールド座標系

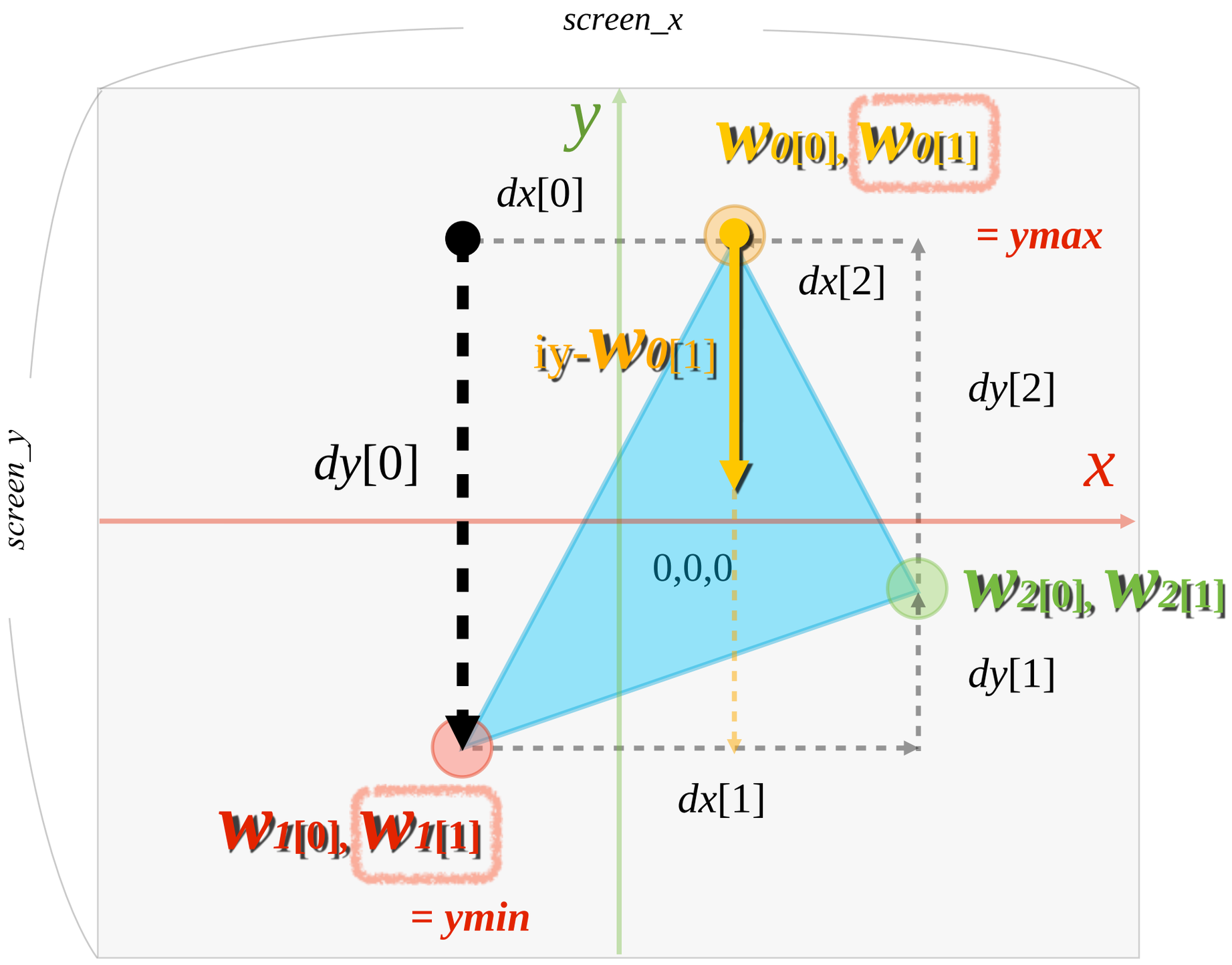



```
int[] dx={0,0,0}; int[] dy={0,0,0}; float[] dz={0,0,0};
dx[0]=int(W1[0]-W0[0]); dy[0]=int(W1[1]-W0[1]);
dx[1]=int(W2[0]-W1[0]); dy[1]=int(W2[1]-W1[1]);
dx[2]=int(W0[0]-W2[0]); dy[2]=int(W0[1]-W2[1]);
dz[0]=Depth[1]-Depth[0];
dz[1]=Depth[2]-Depth[1];
dz[2]=Depth[0]-Depth[2];
```

スクリーン座標系

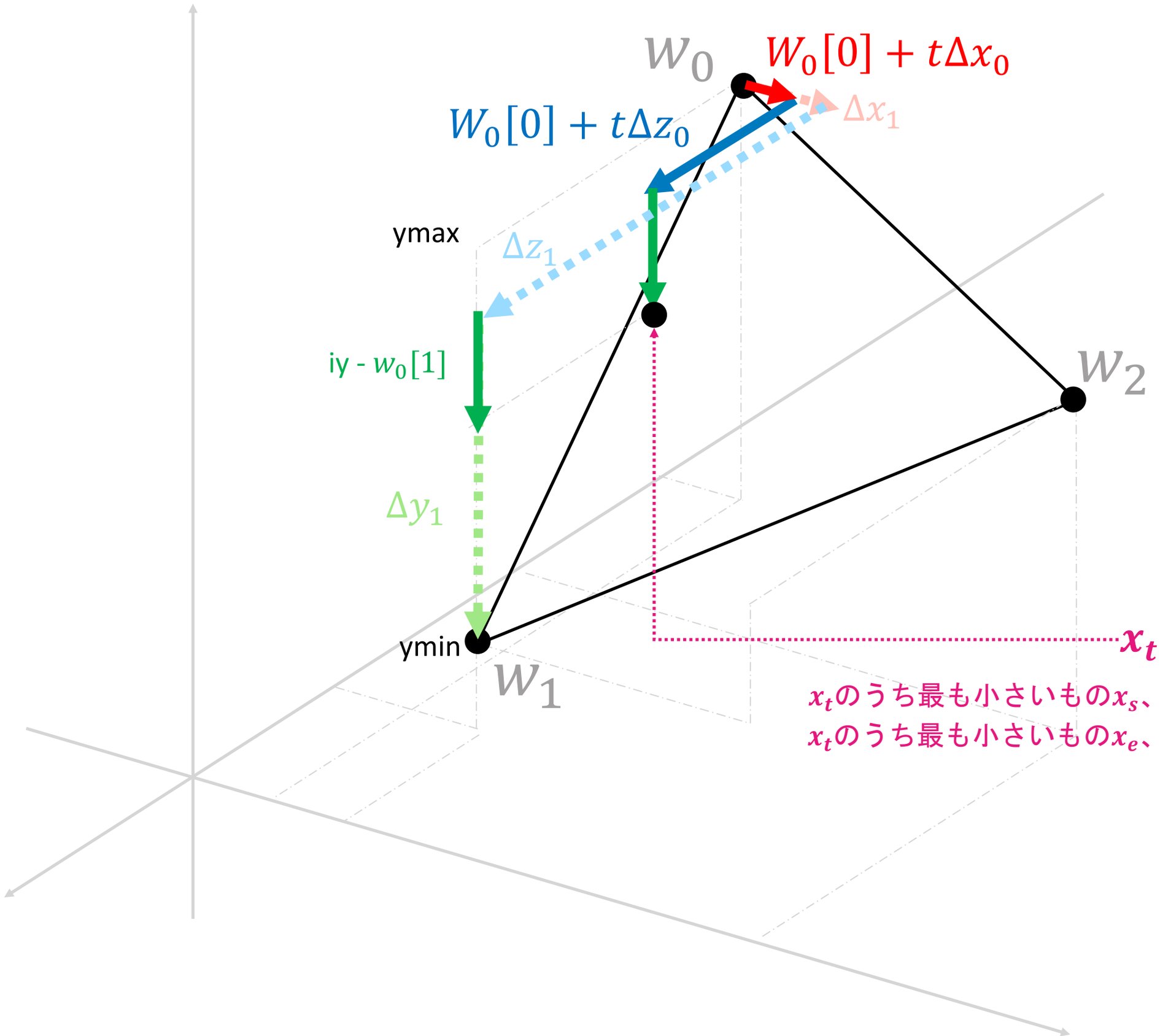
```
for (int iy= ymin; iy<= ymax; iy++) {
```

y方向に走査中...




```
t=(iy-W0[1])/dy[0];
```

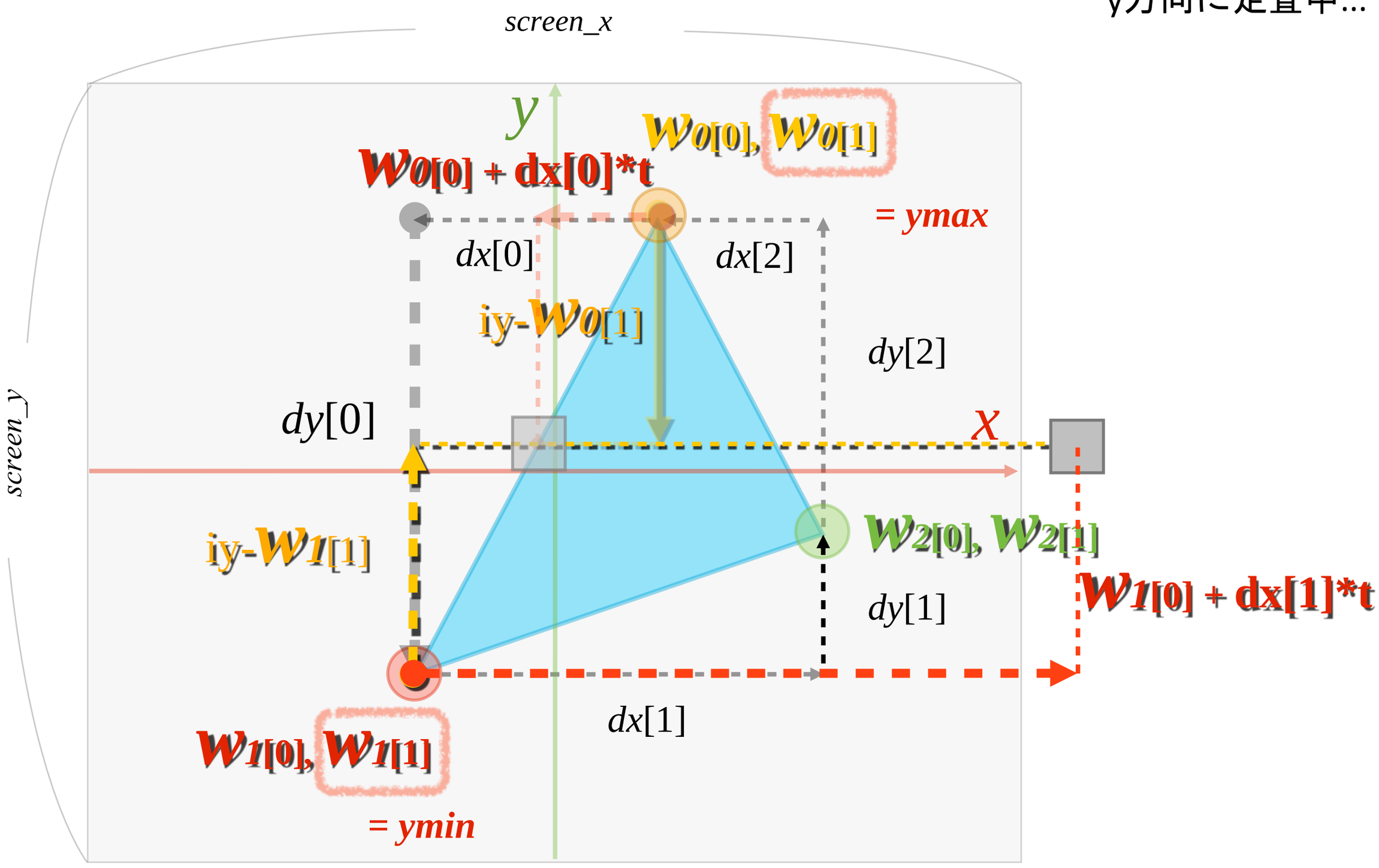
$t = 1$ を超えていないので、三角形の内側にあります。


w_0
$W_0[0] + t\Delta x_0$
$W_0[0] + t\Delta z_0$
Δx_1
ymax
Δz_1
iy - $w_0[1]$
w_2
Δy_1
ymin
w_1
x_tとz_tを計算する
x_tのうち最も小さいものx_s、そのときのx_tをz_sとする
x_tのうち最も小さいものx_e、そのときのx_tをz_eとする

スクリーン座標系

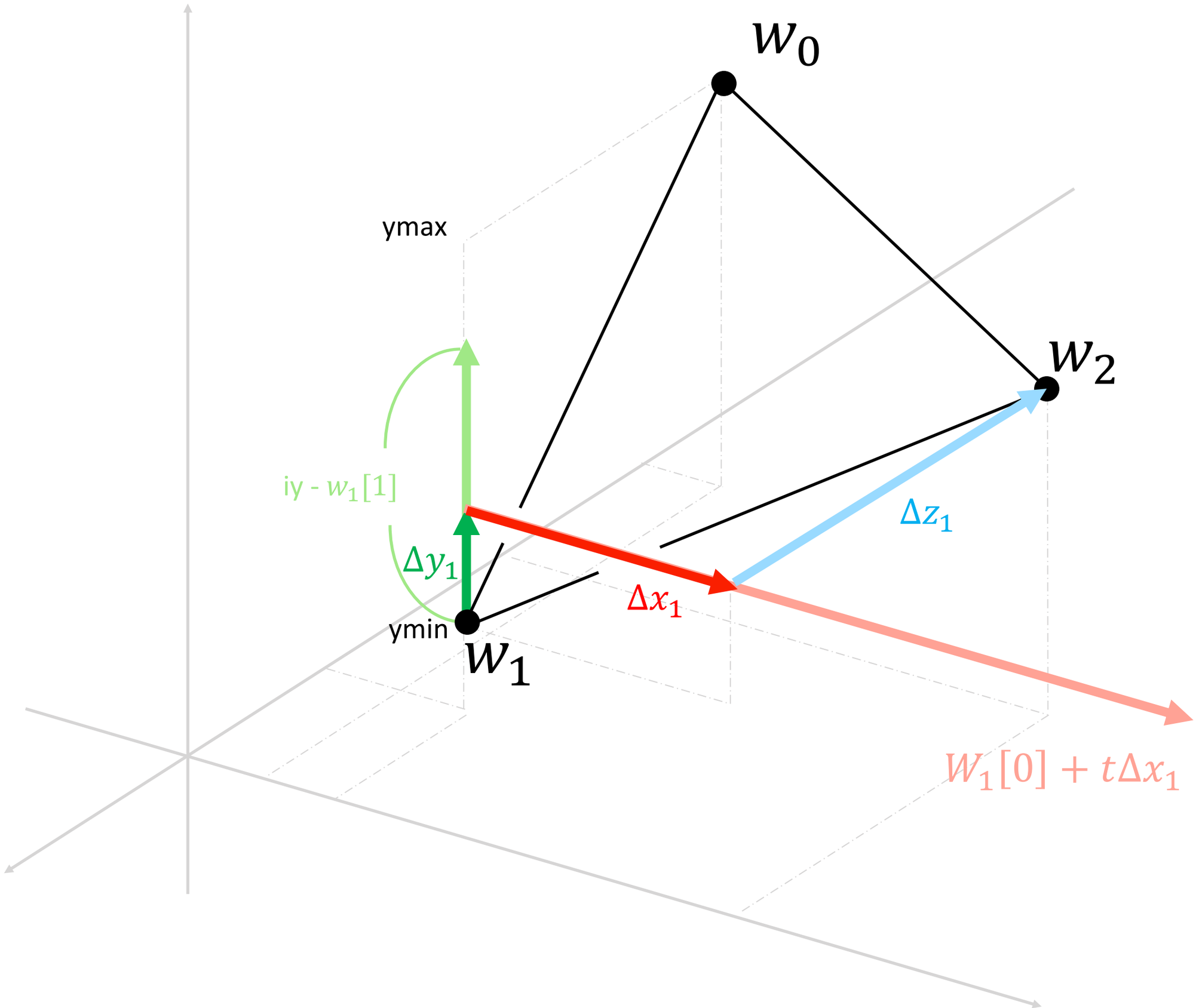
```
for (int iy= ymin; iy<= ymax; iy++) {
```

y方向に走査中...



$t = 1$ を超えると、三角形からはみ出てしまっています。

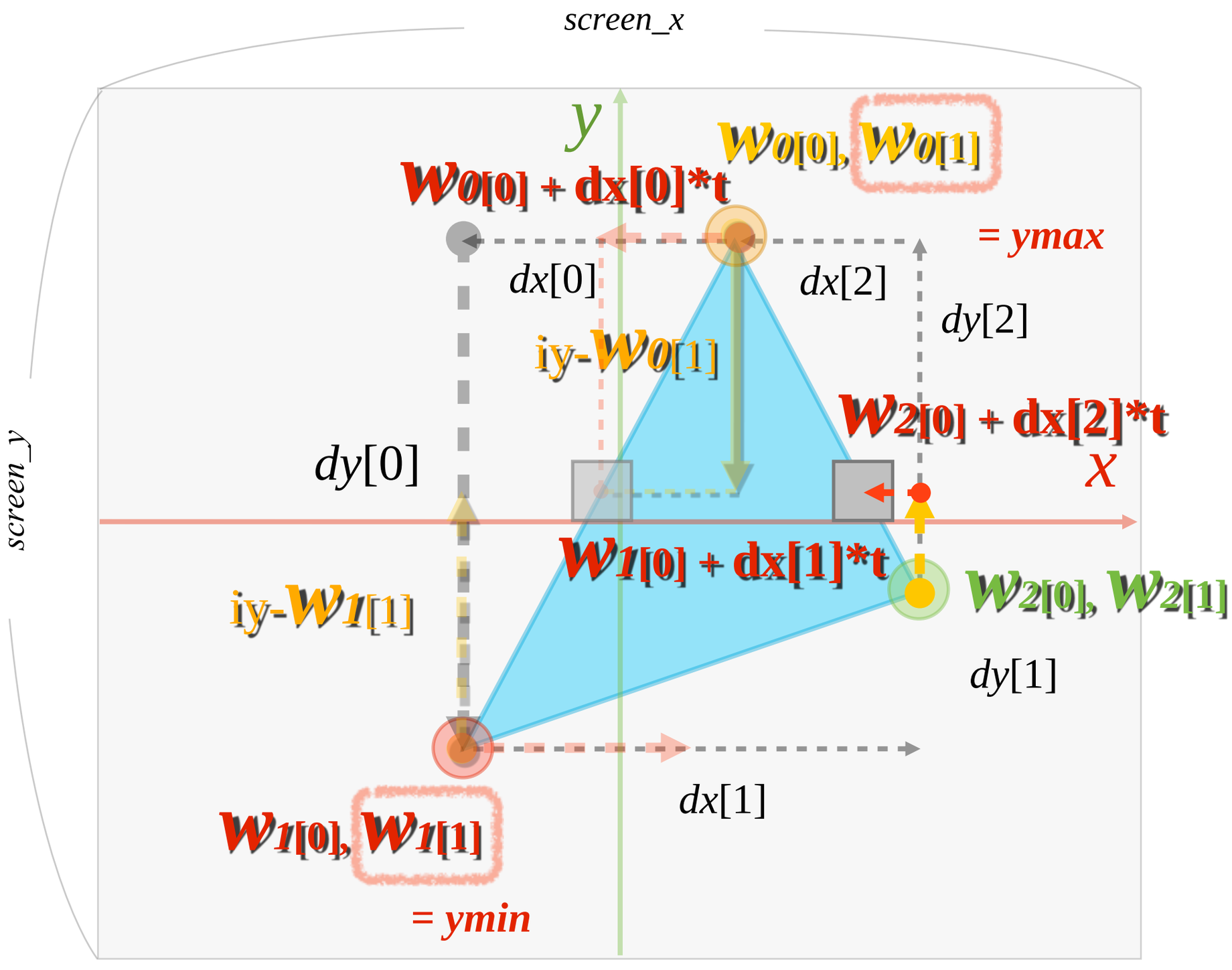
```
if(t>0.0&&t<=1.0)  {  // 0 < t <= 1 のときだけ考えます。
```


w_0
ymax
w_2
iy - $w_1[1]$
Δz_1
Δy_1
Δx_1
ymin
w_1
$W_1[0] + t\Delta x_1$

スクリーン座標系

```
for (int iy=ymin; iy<=ymax; iy++) {
```

y方向に走査中...



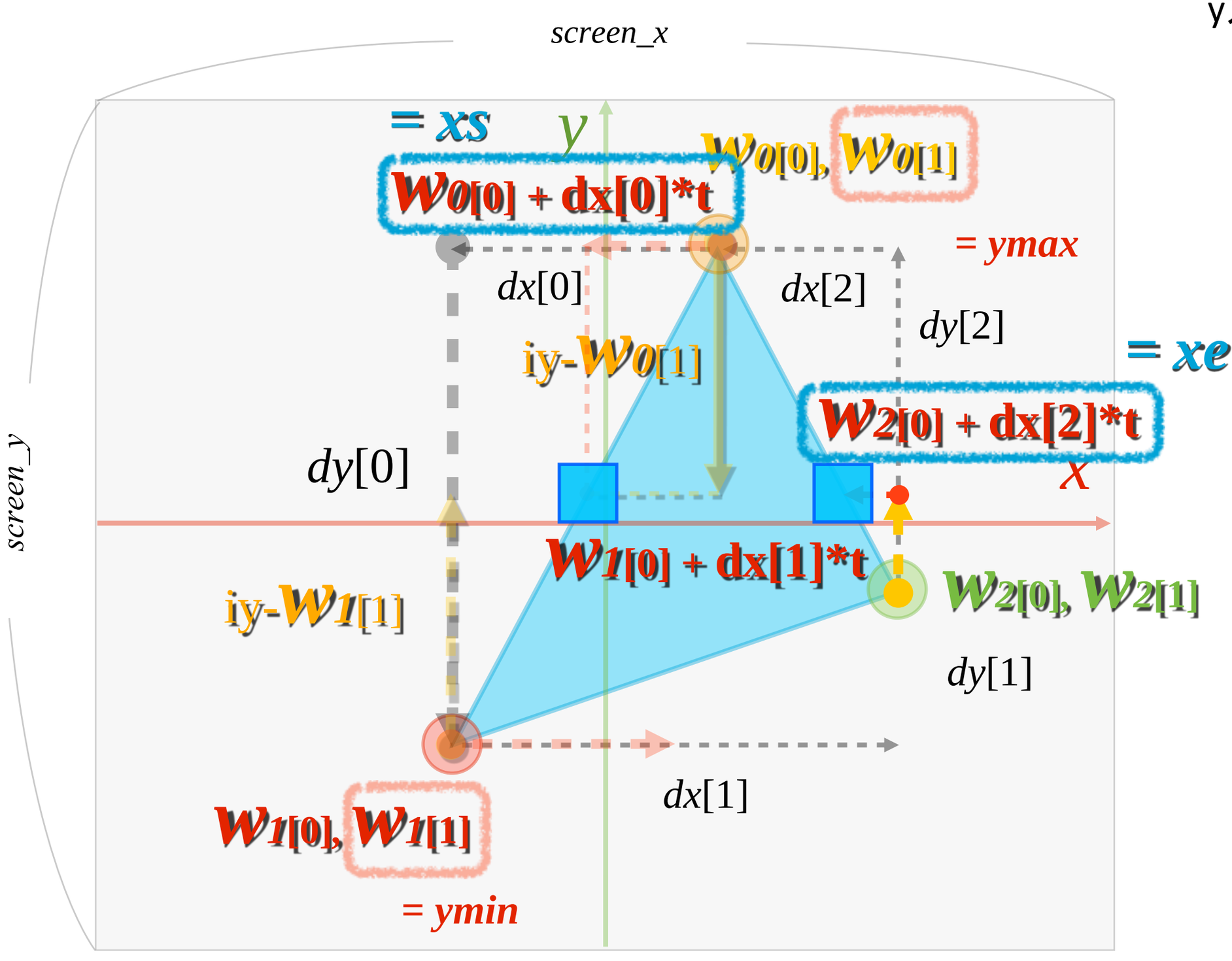

```
t=(iy-W2[1])/(float)dy[2];
```

$t = 1$ を超えていないので、三角形の内側にあります。

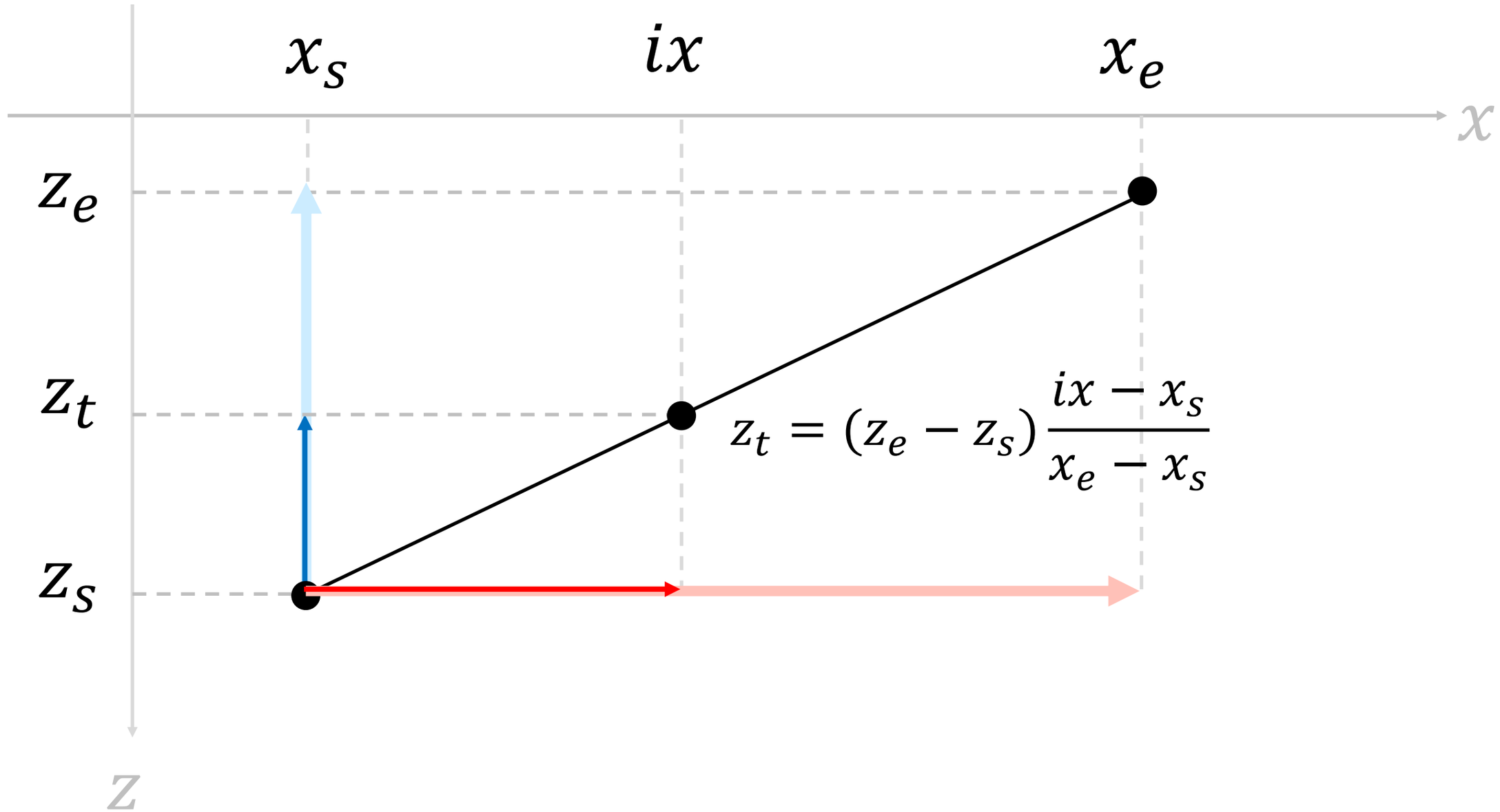
スクリーン座標系

```
for (int iy=ymin; iy<=ymax; iy++) {
```

y方向に走査中...




```
for (int ix=xs ; ix<=xe; ix++) {
    // pixel(ix, iy)を塗りつぶしていく
}
```



z_tの値をz_bufferに保存しておき、
もしより手前のz_tが来た場合は、描画する

```
// use Z-buffer
if(use_z_buffer) {
  if( zt>z_buffer[iy][ix] ) {
    z_buffer[iy][ix]=zt;
    if(ix==xs||ix==xe) {
      image_out[0][iy][ix]
     =image_out[1][iy][ix]
     =image_out[2][iy][ix]=255;
    } else {
      image_out[0][iy][ix]=R; // R
      image_out[1][iy][ix]=G; // G
      image_out[2][iy][ix]=B; // B
    }
  }
}
```

// 最も手前のZバッファを更新して

// X方向の両端の場合は、白く
// つまり、稜線に当たる部分は
// 別の色にして…

// 面に当たる部分は、引数で
// 指定された色を指定します。