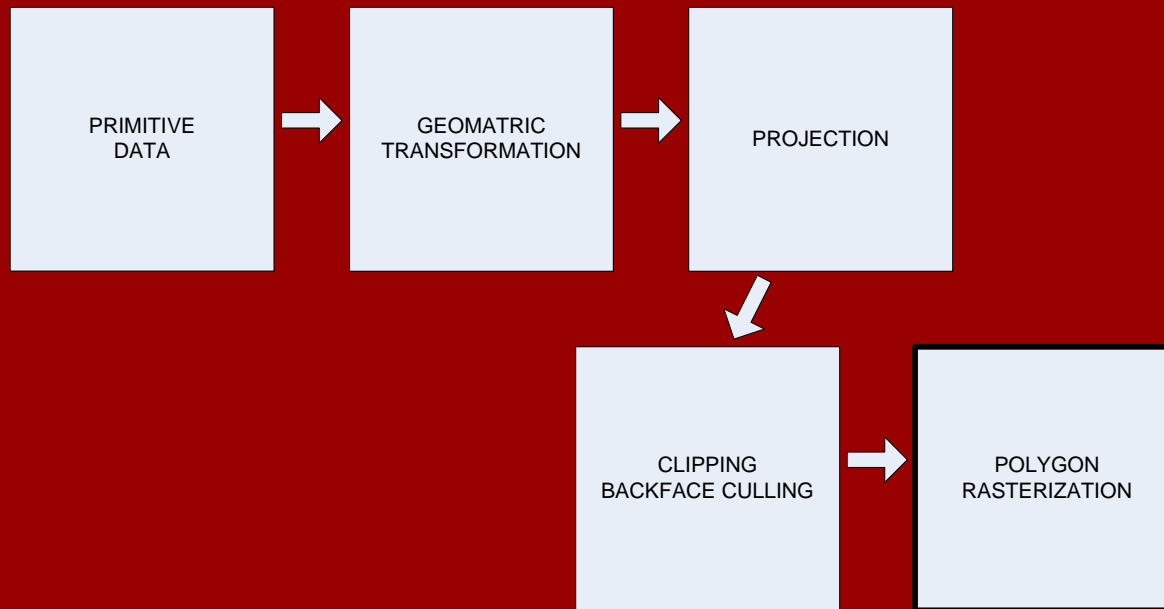

Understanding Polygon Rendering

(noerror@hitel.net)

<http://www.digibath.com/noerror>

2004.08.08

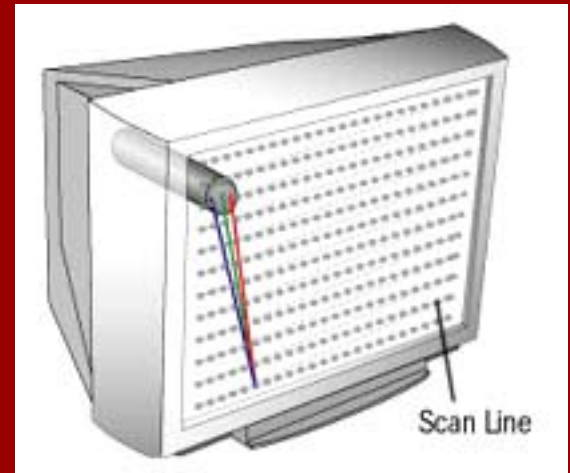
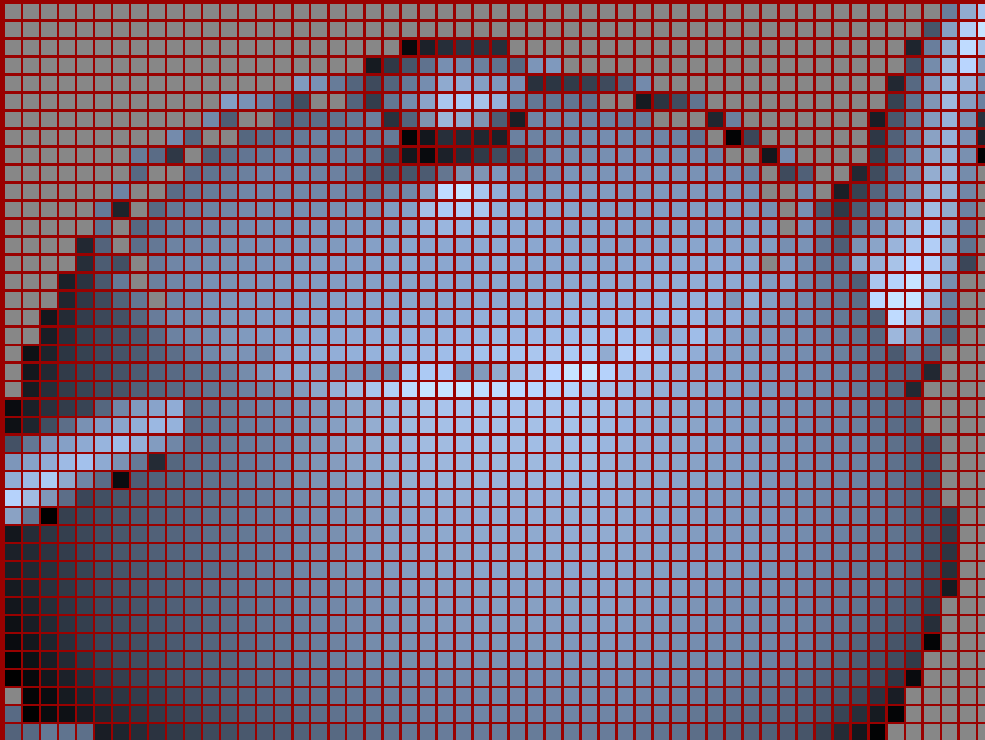
Position in Rendering Pipeline



Rasterization

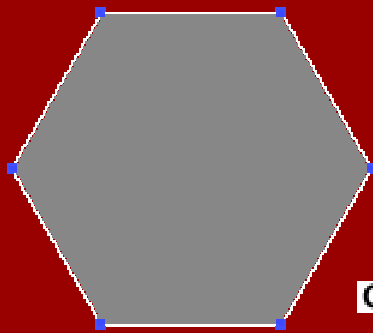
❖ (Rasterization)

?

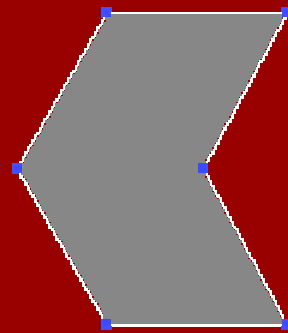


Why Triangle

(Convex Polygon)



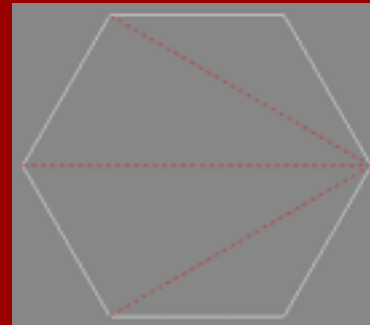
CONVEX



NOT CONVEX

=

(Triangle)



Triangle (I)



3

(Vertex), 3

(Edge)



Y

X



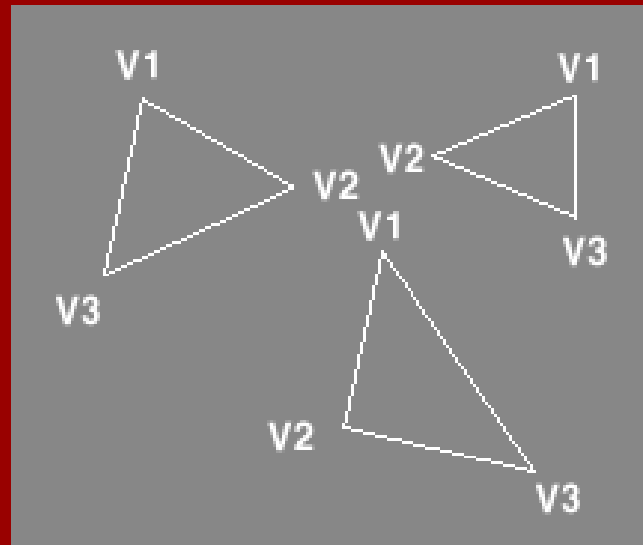
Triangle (II)

❖ Y V1, V2, V3

가

V1 -> V2, V2 -> V3

V1 -> V3



❖ V2

가

Interpolation (I)

❖ (Linear Interpolation)

$$\begin{aligned} F(t) &= F1 * (1 - t) + F2 * t \\ &= F1 + (F2 - F1) * t \end{aligned}$$

❖ 1

```
#include <stdio.h>
void Interpolation(int x1, int x2, int len)
{
    int i;
    for(i=0; i<len; i++)
        printf("%d ", x1 + (x2-x1) * i / len);
}
void main()
{
    Interpolation(5, 20, 20);
}
```



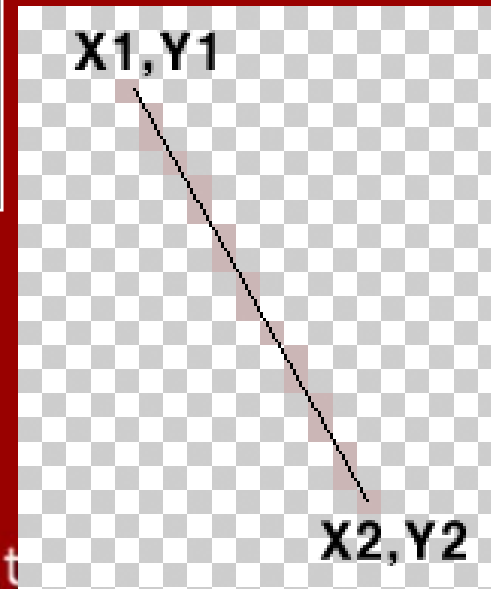
```
D:\WORK\LAB\Softrender\Interpolation\Debug\Interpolation.exe
5 5 6 7 8 8 9 10 11 11 12 13 14 14 15 16 17 17 18 19
```

Interpolation (II)



2 -

```
void DrawLine(int x1, int y1, int x2, int y2, unsigned long color)
{
    if (y1 > y2)
        DrawLine(x2, y2, x1, y1, color);
    else
    {
        int y;
        for(y=y1; y<=y2; y++)
            PutPixel(x1 + (int)((x2 - x1) * (y-y1+0.5f) / (y2-y1+1)), y, color);
    }
}
```



Triangle rendering Sample

```
#include <windows.h>

struct _VTX {
    int x, y;
};

void DrawLine(HDC hdc, int x1, int x2, int y);
void DrawTriangle(HDC hdc, const _VTX *v1, const _VTX *v2, const _VTX *v3);

LRESULT CALLBACK WinProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    static _VTX v[3];
    static int vn=0;
    switch (message)
    {
        case WM_LBUTTONDOWN:
            v[vn%3].x = LOWORD(lParam);
            v[vn%3].y = HIWORD(lParam);
            vn++;
            InvalidateRect(hWnd, NULL, TRUE);
            UpdateWindow(hWnd);
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;
    }
}
```

```

case WM_PAINT:
    {
        PAINTSTRUCT ps;
        HDC hdc;
        hdc = BeginPaint(hWnd, &ps);
        if (vn >= 3)
            DrawTriangle(hdc, &v[0], &v[1], &v[2]);
        EndPaint(hWnd, &ps);
        break;
    }
default:
    return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
}

int APIENTRY WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int
nCmdShow)
{
    HWND hwnd;
    WNDCLASS wcl;
    MSG msg;

    wcl.style = 0;
    wcl.lpfnWndProc = WinProc;
    wcl.cbClsExtra = 0;
    wcl.cbWndExtra = 0;
    wcl.hInstance = hInstance;
    wcl.hIcon = LoadIcon(NULL, (LPCTSTR)IDI_APPLICATION);
    wcl.hCursor = LoadCursor(NULL, IDC_ARROW);

```

```

wcl.hbrBackground = (HBRUSH) GetStockObject(BLACK_BRUSH);
wcl.lpszMenuName = NULL;
wcl.lpszClassName = "_simple";

if (!RegisterClass(&wcl))
    return NULL;

hwnd = CreateWindow(wcl.lpszClassName, "_triangle",
    WS_OVERLAPPEDWINDOW|WS_VISIBLE, CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT, NULL, hInstance, NULL);

while(GetMessage(&msg, NULL, 0, 0)) {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
}

return msg.wParam;
}

void DrawLine(HDC hdc, int x1, int x2, int y)
{
    if (x1 > x2)
    {
        DrawLine(hdc, x2, x1, y);
    } else
    {
        int x;
        for(x=x1; x<=x2; x++)
            ::SetPixel(hdc, x, y, RGB(255, 255, 255));
    }
}

```

```

void DrawTriangle(HDC hdc, const _VTX *v1, const _VTX *v2, const _VTX *v3)
{
    if (v1->y > v2->y) {
        DrawTriangle(hdc, v2, v1, v3);
    } else
    if (v1->y > v3->y) {
        DrawTriangle(hdc, v3, v2, v1);
    } else
    if (v2->y > v3->y) {
        DrawTriangle(hdc, v1, v3, v2);
    } else {
        int y;
        int x1, x2;

        for(y=v1->y; y<v2->y; y++) {
            x1 = v1->x + (int)((v2->x - v1->x) * (y - v1->y + 0.5f) / (v2->y - v1->y));
            x2 = v1->x + (int)((v3->x - v1->x) * (y - v1->y + 0.5f) / (v3->y - v1->y + 1));

            DrawLine(hdc, x1, x2, y);
        }
        for(; y<=v3->y; y++) {
            x1 = v2->x + (int)((v3->x - v2->x) * (y - v2->y + 0.5f) / (v3->y - v2->y + 1));
            x2 = v1->x + (int)((v3->x - v1->x) * (y - v1->y + 0.5f) / (v3->y - v1->y + 1));

            DrawLine(hdc, x1, x2, y);
        }
    }
}

```

Gouraud Shading

- ❖ (vertex) (color)
- ❖ R,G,B
- ❖ (span) Edge R, G,
B

Triangle rendering Sample

Color Blending

```
struct _VTX {
    int x, y;
    unsigned long color;
};

struct _EDGE {
    int x;
    float r, g, b;
};

LRESULT CALLBACK WinProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    ....
    case WM_LBUTTONDOWN:
        v[vn%3].x = LOWORD(lParam);
        v[vn%3].y = HIWORD(lParam);
        v[vn%3].color = RGB(rand()&0xFF, rand()&0xFF, rand()&0xFF);
        vn++;
        InvalidateRect(hWnd, NULL, TRUE);
        UpdateWindow(hWnd);
        break;
    ....
}
```

```

#define _R(C) (float)((C>>16)&0xFF)
#define _G(C) (float)((C>>8)&0xFF)
#define _B(C) (float)(C&0xFF)

void DrawTriangle(HDC hdc, const _VTX *v1, const _VTX *v2, const _VTX *v3)
{
    if (v1->y > v2->y) {
        DrawTriangle(hdc, v2, v1, v3);
    } else
    if (v1->y > v3->y) {
        DrawTriangle(hdc, v3, v2, v1);
    } else
    if (v2->y > v3->y) {
        DrawTriangle(hdc, v1, v3, v2);
    } else {
        int y;
        _EDGE e1, e2;
        float t;

        for(y=v1->y; y<v2->y; y++) {
            t = (y - v1->y + 0.5f) / (v2->y - v1->y);
            e1.x = v1->x + (int) ((v2->x - v1->x) * t);
            e1.r = _R(v1->color) + (_R(v2->color) - _R(v1->color)) * t;
            e1.g = _G(v1->color) + (_G(v2->color) - _G(v1->color)) * t;
            e1.b = _B(v1->color) + (_B(v2->color) - _B(v1->color)) * t;

            t = (y - v1->y + 0.5f) / (v3->y - v1->y + 1);
            e2.x = v1->x + (int) ((v3->x - v1->x) * t);
            e2.r = _R(v1->color) + (_R(v3->color) - _R(v1->color)) * t;
            e2.g = _G(v1->color) + (_G(v3->color) - _G(v1->color)) * t;
            e2.b = _B(v1->color) + (_B(v3->color) - _B(v1->color)) * t;

            DrawLine(hdc, &e1, &e2, y);
        }
    }
}

```

```

for(; y<=v3->y; y++) {
    t = (y - v2->y + 0.5f) / (v3->y - v2->y + 1);
    e1.x = v2->x + (int) ((v3->x - v2->x) * t);
    e1.r = _R(v2->color) + (_R(v3->color) - _R(v2->color)) * t;
    e1.g = _G(v2->color) + (_G(v3->color) - _G(v2->color)) * t;
    e1.b = _B(v2->color) + (_B(v3->color) - _B(v2->color)) * t;

    t = (y - v1->y + 0.5f) / (v3->y - v1->y + 1);
    e2.x = v1->x + (int) ((v3->x - v1->x) * t);
    e2.r = _R(v1->color) + (_R(v3->color) - _R(v1->color)) * t;
    e2.g = _G(v1->color) + (_G(v3->color) - _G(v1->color)) * t;
    e2.b = _B(v1->color) + (_B(v3->color) - _B(v1->color)) * t;

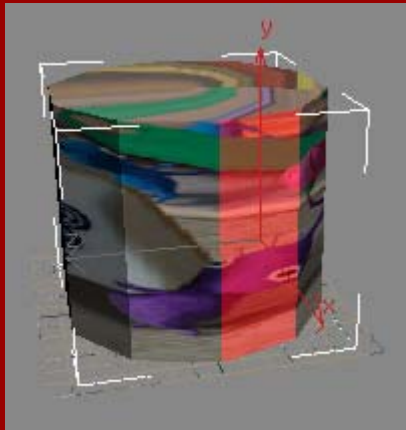
    DrawLine(hdc, &e1, &e2, y);
}
}

void DrawLine(HDC hdc, const _EDGE *e1, const _EDGE *e2, int y)
{
    if (e1->x > e2->x) {
        DrawLine(hdc, e2, e1, y);
    } else {
        int x;
        float r, g, b;
        for(x=e1->x; x<=e2->x; x++) {
            r = e1->r + (e2->r - e1->r) * (x - e1->x + 0.5f) / (e2->x - e1->x + 1);
            g = e1->g + (e2->g - e1->g) * (x - e1->x + 0.5f) / (e2->x - e1->x + 1);
            b = e1->b + (e2->b - e1->b) * (x - e1->x + 0.5f) / (e2->x - e1->x + 1);
            ::SetPixel(hdc, x, y, RGB((int)r, (int)g, (int)b));
        }
    }
}

```

Texture Mapping I

❖ (texture) (mapping)

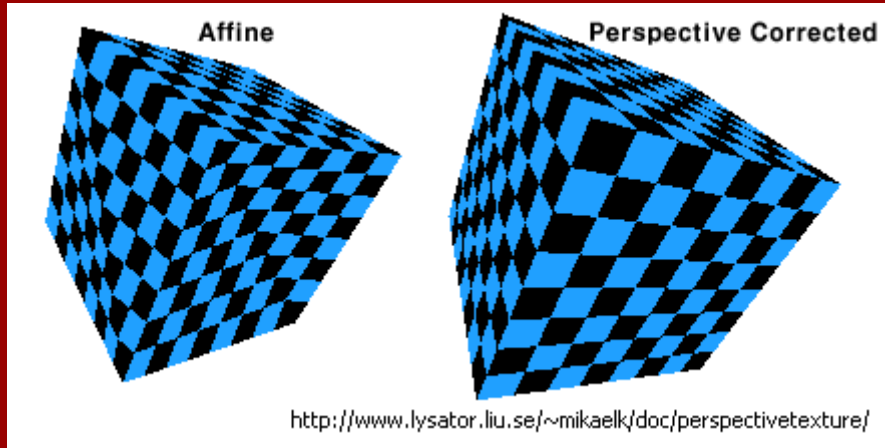


❖ U, V (X, Y)

Perspective Correction (I)



z



z

1/z

Perspective Correction (II)

❖ (Perspective Correct)

$$\text{❖ } F(t) = \frac{F1 * (1-t) / z1 + F2 * t / z2}{(1-t) / z1 + t / z2}$$

❖ N , N u, v
가

Texture Blending (I)

❖ Point Sampling

$$F_{po}(u, v) = \text{Tex}(u', v')$$

$$u' = u * W_{\text{tex}}, v' = v * H_{\text{tex}}$$

❖ Bilinear Filtering

$$F_{bi}(u, v) = I(u', v') * w_{00} + I(u'+1, v') * w_{10} + \\ I(u', v'+1) * w_{01} + I(u'+1, v'+1) * w_{11}$$

$$u'' = u' - \text{floor}(u'), v'' = v' - \text{floor}(v')$$

$$w_{00} = (1-u'') * (1-v''), w_{10} = u'' * (1-v'')$$

$$w_{01} = (1-u'') * v'', w_{11} = u'' * v''$$



Texture Blending (II)

- ❖ Trilinear filtering
 - Bilinear filtering + Mip Map
- ❖ Anisotropic filtering
- ❖ Mip mapping

❖ Z Buffering

Z , W
Z



Issue (I)

❖ Fixed Point Accelation

```
#define INT2FIXED(A)    ((A)<<16)
#define FLOAT2FIXED(A) (int)((A)*65536.0f)
#define FIXED2INT(A)   ((A)>>16)
#define _FIXED int

void Interpolation(int x1, int x2, int len)
{
    int i;
    _FIXED x, dx;
    x = INT2FIXED(x1);
    dx = (INT2FIXED(x2) - INT2FIXED(x1)) / len;
    for(i=0; i<len; i++, x+=dx)
        printf("%d ", FIXED2INT(x));
}
```

,

Issue (II)

❖ Fixed Point Tip -

```
#include <stdio.h>

void Interpolation(int u1, int v1, int u2, int v2, int len) // u, v = 0.0 ~ 127.0
{
    int i;
    int uv, duv;
    uv = ((u1 << 8) & 0x00007FFF) | (((v1 << 8) << 16) & 0x7FFF0000);
    duv = (((u2 - u1) << 8) / len) & 0x00007FFF |
        (((v2 - v1) << 8) / len) << 16) & 0x7FFF0000);
    for(i=0; i<=len; i++, uv=(uv+duv)&0x7FFF7FFF)
    {
        printf("%2d : %d %d \n", i, (uv>>8)&0x7F, (uv>>(16+8))&0x7F);
    }
}

void main()
{
    Interpolation(10, 127, 100, 20, 10);
}
```

Issue (III)

- ❖ Span-buffer Rendering
- ❖ Texture Caching
- ❖ Skipping Zbuffer Clearing

Q/A
