# CS 580 – Discussion
# Hw2 Rasterization
## Week 2

*Lixing Liu*

lixing.liu@usc.edu

# Reminder: HW1

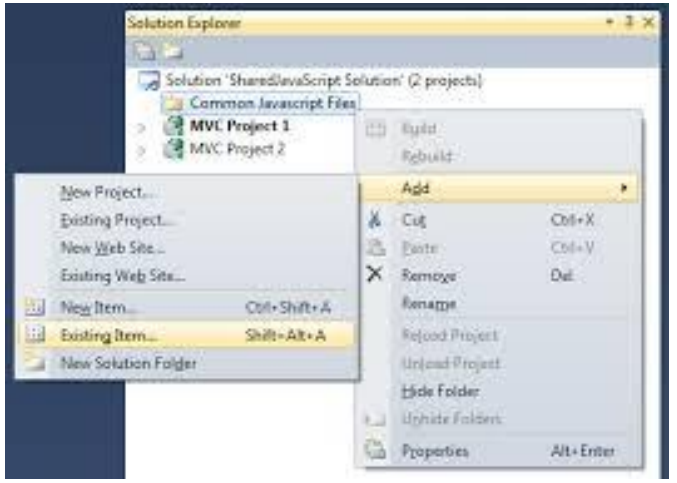
- Due Tue, Aug 28

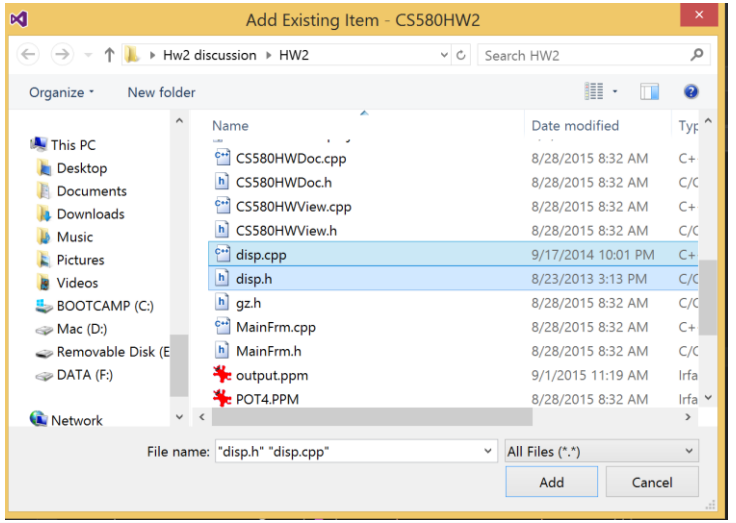
- Note: if you installed VS 2015 Update 2, you'll have .VC.db file (e.g., "CS580HW1.VC.db") instead of .sdf file (e.g., "CS580HW1.sdf")

  * Please delete those files before submitting your assignment to BB

# How to compile Hw2

1. Download Hw2.zip from blackboard
2. Unzip the file and Open .dsw file
3. Copy your functions in <rend.cpp & rend.h> from HW1
4. Implement your new functions
5. Build Solution




Your renderer **will link with your display code from HW1**.
If your display code is not yet correct, you'll have to complete it since you need it now.

POT4.PPM

# Input file

- #define INFILE2  "pot4.screen.asc“

X,  Y,  Z,  Nx,  Ny,  Nz,  U,  V

pot4.screen.asc

```
triangle
193.982361  62.773956  1848056576.000000  -0.238949  -0.249727  -0.293050  0.000000  0.000000
171.412766  39.927429  1842233344.000000  -0.126008  -0.361118  -0.259076  0.000000  0.000000
174.741913  37.348049  1840171392.000000  0.041208  -0.240400  -0.395986  0.000000  0.000000
triangle
193.982361  62.773956  1848056576.000000  -0.238949  -0.249727  -0.293050  0.000000  0.000000
174.741913  37.348049  1840171392.000000  0.041208  -0.240400  -0.395986  0.000000  0.000000
197.087006  60.080017  1845992064.000000  -0.088007  -0.120045  -0.431304  0.000000  0.000000
triangle
197.087006  60.080017  1845992064.000000  -0.088007  -0.120045  -0.431304  0.000000  0.000000
174.741913  37.348049  1840171392.000000  0.041208  -0.240400  -0.395986  0.000000  0.000000
175.644104  35.788799  1840550656.000000  0.317259  0.281337  -0.189629  0.000000  0.000000
```

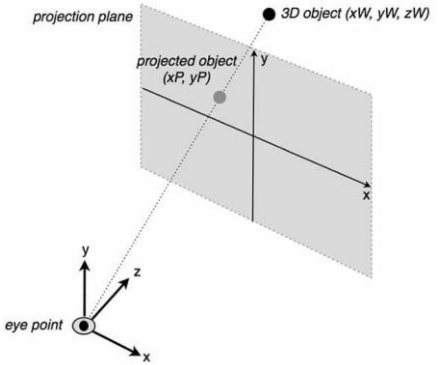
Application2.cpp

```
while( fscanf(infile, "%s", dummy) == 1) {  /* read i
    fscanf(infile, "%f %f %f %f %f %f %f %f",
    &(vertexList[0][0]), &(vertexList[0][1]),
    &(vertexList[0][2]),
    &(normalList[0][0]), &(normalList[0][1]),
    &(normalList[0][2]),
    &(uvList[0][0]), &(uvList[0][1]) );
    fscanf(infile, "%f %f %f %f %f %f %f %f",
    &(vertexList[1][0]), &(vertexList[1][1]),
    &(vertexList[1][2]),
    &(normalList[1][0]), &(normalList[1][1]),
    &(normalList[1][2]),
    &(uvList[1][0]), &(uvList[1][1]) );
    fscanf(infile, "%f %f %f %f %f %f %f %f",
    &(vertexList[2][0]), &(vertexList[2][1]),
    &(vertexList[2][2]),
    &(normalList[2][0]), &(normalList[2][1]),
    &(normalList[2][2]),
    &(uvList[2][0]), &(uvList[2][1]) );
```

v1

v2

v3




- Tris are pre-transformed into screen coordinates for HW2

- Input file has tris with X,Y,Z ready for rasterization


# Input file

- #define INFILE2  "pot4.screen.asc“

X,  Y,  Z,  Nx,  Ny,  Nz,  U,  V

pot4.screen.asc

```
triangle
193.982361   62.773956   1848056576.000000   -0.238949   -0.249727   -0.293050   0.000000   0.000000
171.412766   39.927429   1842233344.000000   -0.126008   -0.361118   -0.259076   0.000000   0.000000
174.741913   37.348049   1840171392.000000   0.041208   -0.240400   -0.395986   0.000000   0.000000
triangle
193.982361   62.773956   1848056576.000000   -0.238949   -0.249727   -0.293050   0.000000   0.000000
174.741913   37.348049   1840171392.000000   0.041208   -0.240400   -0.395986   0.000000   0.000000
197.087006   60.080017   1845992064.000000   -0.088007   -0.120045   -0.431304   0.000000   0.000000
triangle
197.087006   60.080017   1845992064.000000   -0.088007   -0.120045   -0.431304   0.000000   0.000000
174.741913   37.348049   1840171392.000000   0.041208   -0.240400   -0.395986   0.000000   0.000000
175.644104   35.788799   1840550656.000000   0.317259   0.281337   -0.189629   0.000000   0.000000
```

- /* Color is assigned for each tri

```
* Set up shading attributes for each triangle
*/
shade2(normalList[0], color);/* shade based on the norm of vert0 */
valueListColor[0] = (GzPointer)color;
nameListColor[0] = GZ_RGB_COLOR;
GzPutAttribute(m_pRender, 1, nameListColor, valueListColor);
```

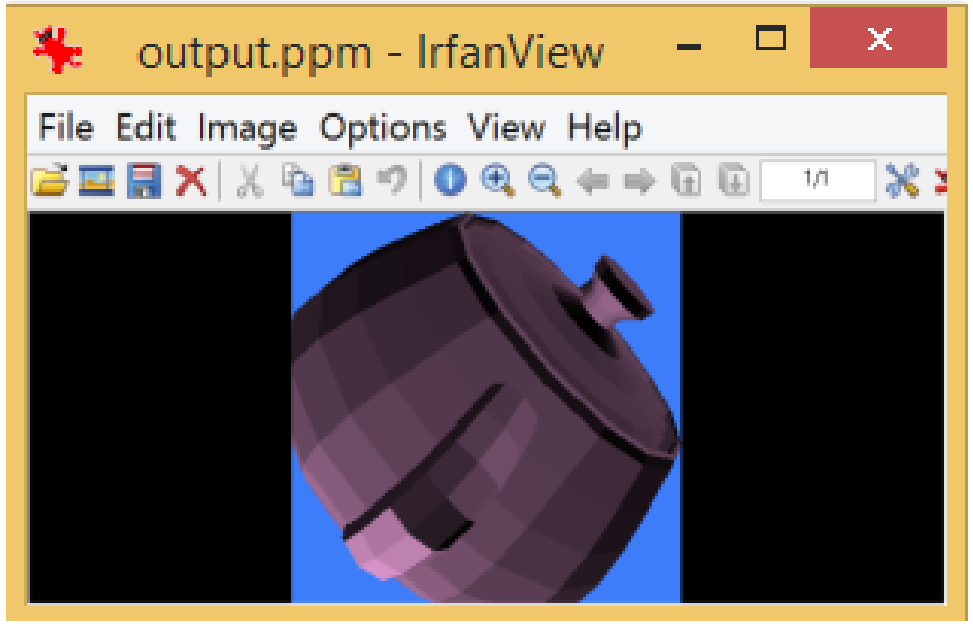
Application2.cpp

```
while( fscanf(infile, "%s", dummy) == 1) {  /* read in tri word */
    fscanf(infile, "%f %f %f %f %f %f %f %f",
    &(vertexList[0][0]), &(vertexList[0][1]),
    &(vertexList[0][2]),
    &(normalList[0][0]), &(normalList[0][1]),
    &(normalList[0][2]),
    &(uvList[0][0]), &(uvList[0][1]) );
    fscanf(infile, "%f %f %f %f %f %f %f %f",
    &(vertexList[1][0]), &(vertexList[1][1]),
    &(vertexList[1][2]),
    &(normalList[1][0]), &(normalList[1][1]),
    &(normalList[1][2]),
    &(uvList[1][0]), &(uvList[1][1]) );
    fscanf(infile, "%f %f %f %f %f %f %f %f",
    &(vertexList[2][0]), &(vertexList[2][1]),
    &(vertexList[2][2]),
    &(normalList[2][0]), &(normalList[2][1]),
    &(normalList[2][2]),
    &(uvList[2][0]), &(uvList[2][1]) );
/*
This doesn't really belong in the application program, but for this
simplified case of a renderer that doesn't do any shading itself, this
is the easiest place to put it.
*/

void shade2(GzCoord norm, GzCoord   color)
{
  GzCoord   light;
  float     coef;

  light[0] = 0.707f;
  light[1] = 0.5f;
  light[2] = 0.5f;

  coef = light[0]*norm[0] + light[1]*norm[1] + light[2]*norm[2];
  if (coef < 0)     coef *= -1;

  if (coef > 1.0)   coef = 1.0;
  color[0] = coef*0.95f;
  color[1] = coef*0.65f;
  color[2] = coef*0.88f;
}
```

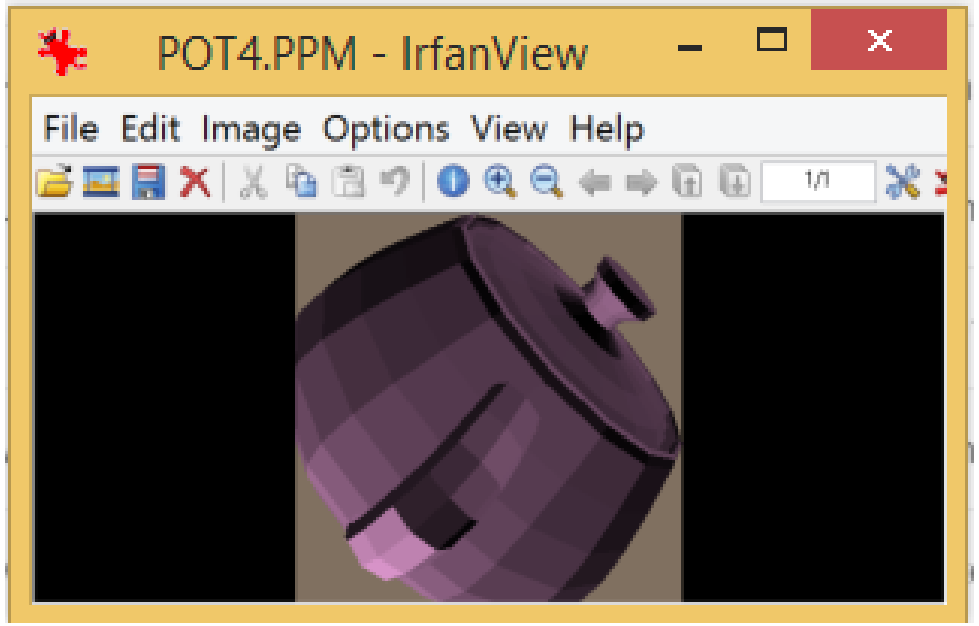
# Output

- The result images are made into a 256x256 window.
- Do not change the resolution/size of your display image since the transformation is precomputed for that image size.

#define OUTFILE2 "output.ppm"




You can compare the result with POT4.PPM



# Render – Overview

- The renderer has to manage the display

Rend.h

```
NDER
NDER
typedef struct {                /* define a renderer */
  GzDisplay     *display;
  GzCamera      camera;
  short     matlevel;   /* top of stack - current xform */
  GzMatrix      Ximage[MATLEVELS];  /* stack of xforms (Xsm) */
  GzMatrix      Xnorm[MATLEVELS];   /* xform for norms (Xim) */
  GzMatrix      Xsp;        /* NDC to screen (pers-to-screen) */
  GzColor       flatcolor;    /* color state for flat shaded triangles */
  int           interp_mode;
  int           numlights;
  GzLight       lights[MAX_LIGHTS];
  GzLight       ambientlight;
  GzColor       Ka, Kd, Ks;
  float     spec;       /* specular power */
  GzTexture     tex_fun;  /* tex_fun(float u, float v, GzColor color) */
}  GzRender;
#endif
```

Application2.cpp

```
status |= GzNewFrameBuffer(&m_pFrameBuffer, m_nWidth, m_nHeight);
status |= GzNewDisplay(&m_pDisplay, m_nWidth, m_nHeight);

status |= GzGetDisplayParams(m_pDisplay, &xRes, &yRes);

status |= GzNewRender(&m_pRender, m_pDisplay);

status |= GzBeginRender(m_pRender);
```

You'll need to implement theses two functions:

```
int GzPutAttribute(GzRender *render, int numAttributes, GzToken *nameList,
    GzPointer *valueList);
int GzPutTriangle(GzRender *render, int numParts, GzToken *nameList,
    GzPointer *valueList);
```

# Render – GzBeginRender()

- GzBeginRender() is called to initialize everything for a new frame

Application1.cpp

```
status |= GzNewFrameBuffer(&m_pFrameBuffer, m_nWidth, m_nHeight);

status |= GzNewDisplay(&m_pDisplay, m_nWidth, m_nHeight);

status |= GzGetDisplayParams(m_pDisplay, &xRes, &yRes);

status |= GzInitDisplay(m_pDisplay);  /* init for new frame */
```

Application2.cpp

```
status |= GzNewFrameBuffer(&m_pFrameBuffer, m_nWidth, m_nHeight);
status |= GzNewDisplay(&m_pDisplay, m_nWidth, m_nHeight);

status |= GzGetDisplayParams(m_pDisplay, &xRes, &yRes);

status |= GzNewRender(&m_pRender, m_pDisplay);

status |= GzBeginRender(m_pRender);
```

rend.h

```
#ifndef GZRENDER
#define GZRENDER
typedef struct {            /* define a renderer */
  GzDisplay     *display;
  GzCamera      camera;
  short     matlevel;   /* top of stack - current xform */
  GzMatrix      Ximage[MATLEVELS];  /* stack of xforms (Xsm) */
  GzMatrix      Xnorm[MATLEVELS];   /* xform for norms (Xim) */
  GzMatrix      Xsp;        /* NDC to screen (pers-to-screen) */
  GzColor       flatcolor;    /* color state for flat shaded triangles */
  int           interp_mode;
  int           numlights;
  GzLight       lights[MAX_LIGHTS];
  GzLight       ambientlight;
  GzColor       Ka, Kd, Ks;
  float     spec;       /* specular power */
  GzTexture     tex_fun;  /* tex_fun(float u, float v, GzColor color) */
}  GzRender;
#endif

// Function declaration
int GzNewRender(GzRender **render, GzDisplay *display);
int GzFreeRender(GzRender *render);
int GzBeginRender(GzRender  *render);
int GzPutAttribute(GzRender *render, int numAttributes, GzToken *nameList,
    GzPointer *valueList);
int GzPutTriangle(GzRender *render, int numParts, GzToken *nameList,
    GzPointer *valueList);
```

# Z-buffering

```
GzInitDisplay(m_pDisplay);
```

```
int GzBeginRender(GzRender  *render)
{
/*
- set up for start of each frame - init frame buffer
*/
    return GZ_SUCCESS;
}
```

- int GzInitDisplay(GzDisplay*display)
  - The renderer must do z-buffering so you need to initialize z in its framebuffer, in addition to setting the background color
  - Initial z-value is MAXINT, the maximum positive integer value
- int GzPutDisplay(GzDisplay *display, int i, int j, GzIntensity r, GzIntensity g, GzIntensity b, GzIntensity a, GzDepth z)
  - If you did not set z value in Hw1

rend.h

```
#ifndef GZRENDER
#define GZRENDER
typedef struct {           /* define a renderer */
  GzDisplay     *display;
  GzCamera      camera;
  short         matlevel;   /* top of stack - current xform */
  GzMatrix      Ximage[MATLEVELS];  /* stack of xforms (Xsm) */
  GzMatrix      Xnorm[MATLEVELS];   /* xform for norms (Xim) */
  GzMatrix      Xsp;        /* NDC to screen (pers-to-screen) */
  GzColor       flatcolor;    /* color state for flat shaded triangles */
  int           interp_mode;
  int           numlights;
  GzLight       lights[MAX_LIGHTS];
  GzLight       ambientlight;
  GzColor       Ka, Kd, Ks;
  float     spec;       /* specular power */
  GzTexture     tex_fun;  /* tex_fun(float u, float v, GzColor color) */
}  GzRender;
#endif
```

```
/* define a display type */
#ifndef GZ_DISPLAY
typedef struct {
  unsigned short    xres;
  unsigned short    yres;
  GzPixel       *fbuf;      /* frame buffer array */
} GzDisplay;
#define GZ_DISPLAY
#endif;
```

```
typedef struct {
  GzIntensity    red;
  GzIntensity    green;
  GzIntensity    blue;
  GzIntensity    alpha;
  GzDepth    z;
} GzPixel;
#define GZ_PIXEL
#endif;
```

# Tokens and Values

- int GzPutAttribute (GzRender *render, int numAttributes,
  GzToken *nameList, GzPointer *valueList)

```
typedef int     GzToken;
typedef void    *GzPointer;
```

- List of Tokens (ints) List of values
  GZ_RGB_COLOR Float Color [3]
  ..... .....
- Only use GZ_RGB_COLOR for HW2, but other data will be passed in later HWs
- Increment through tokens (ints) and use (sizeof) token type to increment the pointer through the value list

gz.h

```
/* renderer-state color values for default (flat) shading */
#define GZ_RGB_COLOR            99  /* we use RGB color space */
```

```
// Function declaration
int GzNewRender(GzRender **render, GzDisplay *display);
int GzFreeRender(GzRender *render);
int GzBeginRender(GzRender *render);
int GzPutAttribute(GzRender *render, int numAttributes, GzToken *nameList,
    GzPointer *valueList);
int GzPutTriangle(GzRender *render, int numParts, GzToken *nameList,
    GzPointer *valueList);
```

Application2.cpp

```
/*
* Set up shading attributes for each triangle
*/
shade2(normalList[0], color);/* shade based on the norm of vert0 */
valueListColor[0] = (GzPointer)color;
nameListColor[0] = GZ_RGB_COLOR;
GzPutAttribute(m_pRender, 1, nameListColor, valueListColor);
```

gz.h (hw6)

```
/* renderer-state default pixel color */
#define GZ_RGB_COLOR            99

#define GZ_INTERPOLATE  95      /* interpolation mode */

#define GZ_DIRECTIONAL_LIGHT    79  /* directional light */
#define GZ_AMBIENT_LIGHT        78  /* ambient light type */

#define GZ_AMBIENT_COEFFICIENT      1001    /* Ka material property */
#define GZ_DIFFUSE_COEFFICIENT      1002    /* Kd material property */
#define GZ_SPECULAR_COEFFICIENT     1003    /* Ks material property */
#define GZ_DISTRIBUTION_COEFFICIENT 1004    /* specular power of material */
```

# Render – GzPutAttribute

- A simple shading function is within the application and it computes a color for each triangle
- Color is sent to renderer via the generic GzPutAttribute() call that uses pointers to a token list and value list
- Color is passed to Renderer as RGB array of floats defined over the range [0.0, 1.0]

render.cpp

```
int GzPutAttribute(GzRender *render, int numAttributes, GzToken *nameList,
    GzPointer *valueList) /* void** valuelist */
{
/*
- set renderer attribute states (e.g.: GZ_RGB_COLOR default color)
- later set shaders, interpolaters, texture maps, and lights
*/

    return GZ_SUCCESS;
}
```

Application2.cpp

```
/*
* Set up shading attributes for each triangle
*/
shade2(normalList[0], color);/* shade based on the norm of vert0 */
[redacted]
nameListColor[0] = GZ_RGB_COLOR;
[redacted]olor, valueListColor);
```

render.h

```
#ifndef GZRENDER
#define GZRENDER
typedef struct {            /* define a renderer */
  GzDisplay     *display;
  GzCamera      camera;
  short     matlevel;   /* top of stack - current xform */
  GzMatrix      Ximage[MATLEVELS];  /* stack of xforms (Xsm) */
  GzMatrix      Xnorm[MATLEVELS];   /* xform for norms (Xim) */
  GzMatrix      Xsp;        /* NDC to screen (pers-to-screen) */
  GzColor       flatcolor;   /* color state for flat shaded triangles */
  int           interp_mode;
  int           numlights;
  GzLight       lights[MAX_LIGHTS];
  GzLight       ambientlight;
  GzColor       Ka, Kd, Ks;
  float     spec;       /* specular power */
  GzTexture     tex_fun;  /* tex_fun(float u, float v, GzColor color) */
}  GzRender;
#endif
```

# Render – GzPutTriangle

- Same as GzPutAttribute function – only use GZ_POSITION token right now – no other triangle data

```
int GzPutTriangle(GzRender *render, int numParts, GzToken *nameList,
    GzPointer *valueList)
/* numParts - how many names and values */
{
/*
- pass in a triangle description with tokens and values corresponding to
      GZ_NULL_TOKEN:        do nothing - no values
      GZ_POSITION:          3 vert positions
- Invoke the scan converter and return an error code
*/

    return GZ_SUCCESS;
}

/*
 * name list tokens
 */
#define GZ_NULL_TOKEN           0   /* triangle vert attributes */
#define GZ_POSITION             1
#define GZ_NORMAL               2
#define GZ_TEXTURE_INDEX        3
```

- Rasterization
  - LEE or Scan Line(DDA)

```
// Function declaration
int GzNewRender(GzRender **render, GzDisplay *display);
int GzFreeRender(GzRender *render);
int GzBeginRender(GzRender  *render);
int GzPutAttribute(GzRender *render, int numAttributes, GzToken *nameList,
    GzPointer *valueList);
int GzPutTriangle(GzRender *render, int numParts, GzToken *nameList,
    GzPointer *valueList);
```

Application2.cpp

```
while( fscanf(infile, "%s", dummy) == 1) {  /* read in tri word */
    fscanf(infile, "%f %f %f %f %f %f %f %f",
    &(vertexList[0][0]), &(vertexList[0][1]),
    &(vertexList[0][2]),
    &(normalList[0][0]), &(normalList[0][1]),
    &(normalList[0][2]),
    &(uvList[0][0]), &(uvList[0][1]) );
    fscanf(infile, "%f %f %f %f %f %f %f %f",
    &(vertexList[1][0]), &(vertexList[1][1]),
    &(vertexList[1][2]),
    &(normalList[1][0]), &(normalList[1][1]),
    &(normalList[1][2]),
    &(uvList[1][0]), &(uvList[1][1]) );
    fscanf(infile, "%f %f %f %f %f %f %f %f",
    &(vertexList[2][0]), &(vertexList[2][1]),
    &(vertexList[2][2]),
    &(normalList[2][0]), &(normalList[2][1]),
    &(normalList[2][2]),
    &(uvList[2][0]), &(uvList[2][1]) );
```

v1

v2

v3

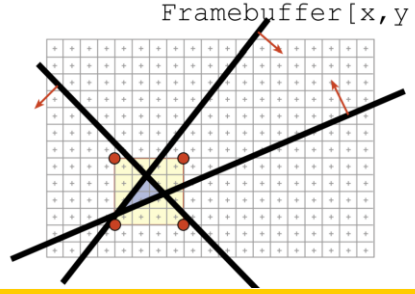
```
/*
 * Set the value pointers to the first vertex of the
 * triangle, then feed it to the renderer
 */
valueListTriangle[0] = (GzPointer)vertexList;

GzPutTriangle(m_pRender, 1, nameListTriangle, valueListTriangle);
```

# LEE or Scan Line(DDA)

- More information for rasterization method
  - http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-837-computer-graphics-fall-2012/lecture-notes/MIT6_837F12_Lec21.pdf

## LEE – P.57

```
For every triangle
   Compute projection for vertices, compute the Ei
   Compute bbox, clip bbox to screen limits
   For all pixels in bbox
      Evaluate edge functions aix + biy + ci
      If all > 0
          Framebuffer[x,y ] = triangleColor
```



## Scan Line – P.68

- Compute the boundary pixels using line rasterization
- Fill the spans