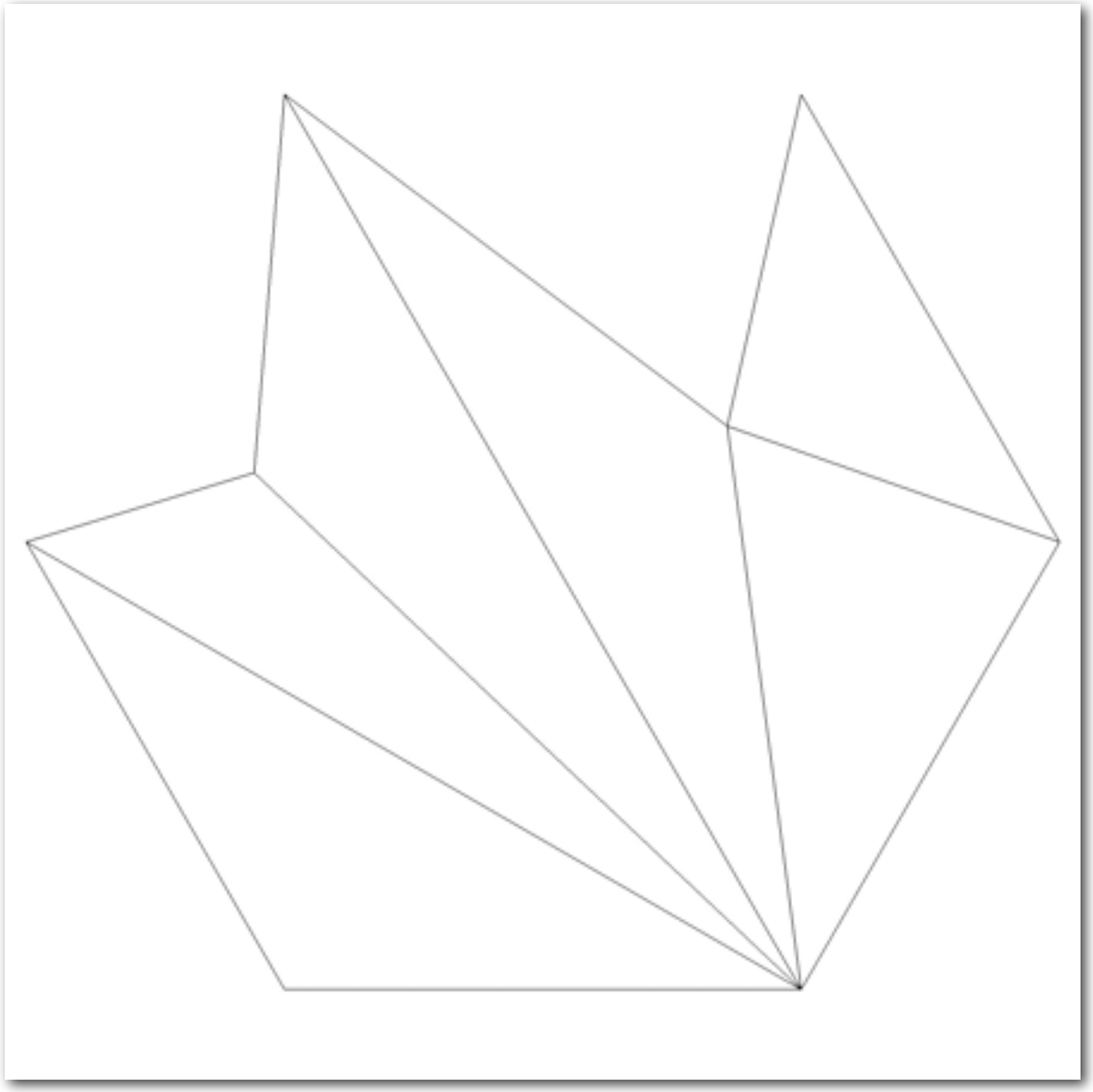


Monash University • Clayton's School of Information Technology

CSE3313 Computer Graphics

Lecture 8: OpenGL — Fonts, Tessellation, Transformations, Errors

Fonts in GLUT

- GLUT provides a few raster and stroke fonts. It is also possible to access the fonts in the windowing system.
- We can access a single character from a **monotype** or evenly spaced font by the function call

```
glutStrokeCharacter(
        GLUT_STROKE_MONO_ROMAN, int character )
```

- Some words of caution:
  - The font may need to be scaled to fit in with the rest of the program output;
  - Calling this function includes a translation to the bottom right of the character box to prepare for the next character.
  - Transformations (scaling, rotation, translation, etc.) affect the OpenGL state so we might need to use `glPushMatrix` and `glPopMatrix` to return to the state before `glPushMatrix` was called.

Drawing Text (cont.)

- As part of the OpenGL state there is a **raster position** which identifies where the next raster primitive will be placed. This can be set via `glRasterPos*( )`, e.g. `glRasterPos2i(10,10)`

- The raster position can be set in world coordinates. The current raster position is updated automatically following a call to `glutBitmapCharacter()`

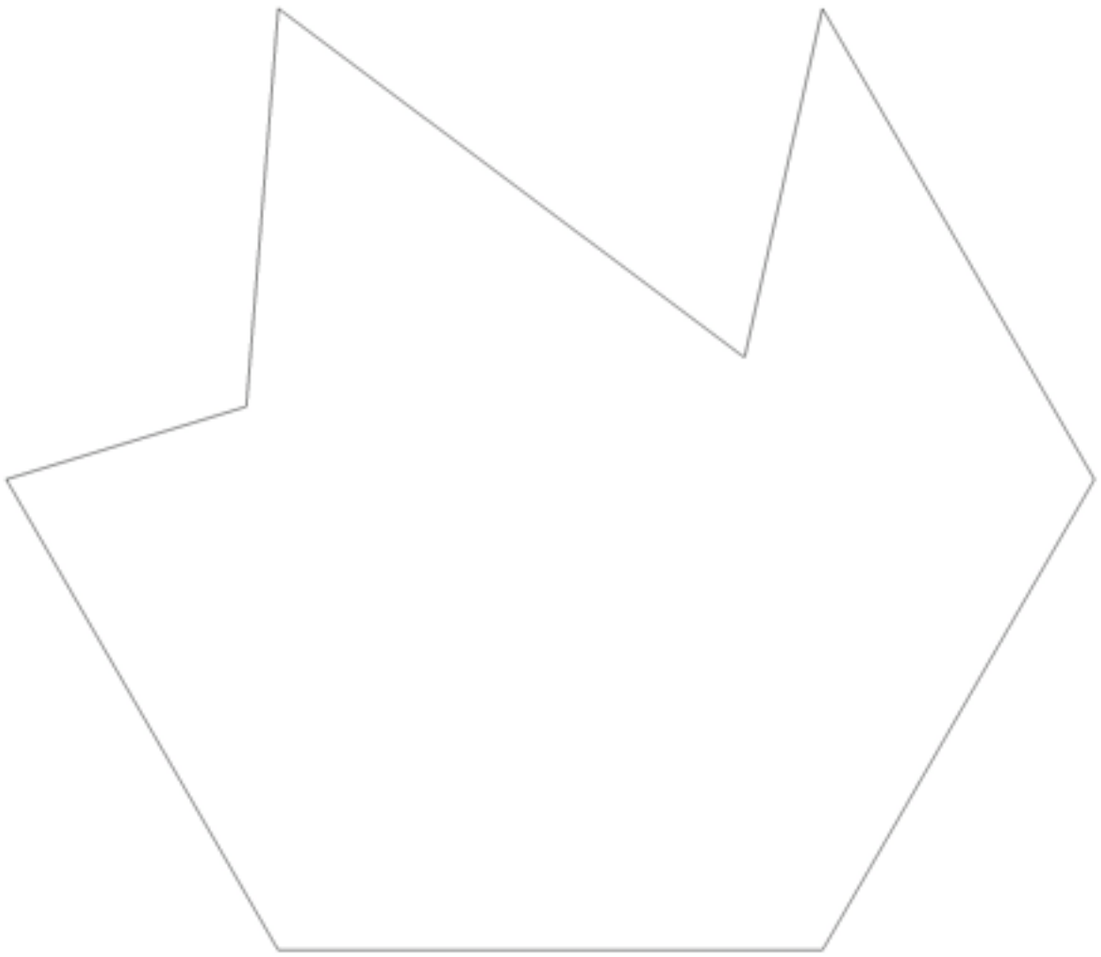
- You can query the width of a bitmap character using
  `int glutBitmapWidth( GLUTbitmapFont font, int char )`
  which returns the width of the character in the font font.

- Example:

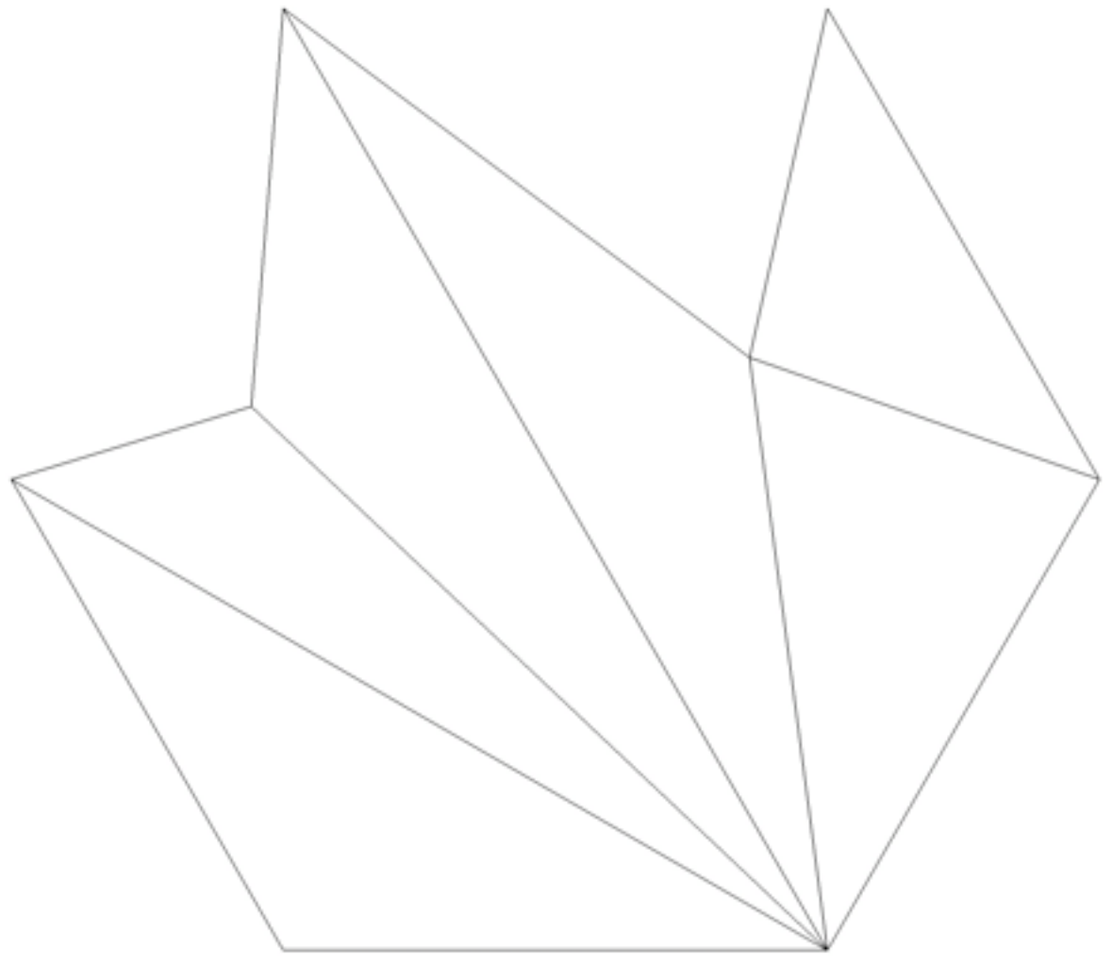
```
char * text = “hello world”; int i;
glRasterPos2i(10,10);
for (i = 0; i < strlen(text); ++i)
    glutBitmapCharacter(GLUT_BITMAP_9_BY_15,
text[i]);
```

Concave Polygons

- `GL_POLYGON` can only render simple, convex polygons. To draw concave polygons we need to use the polygon tessellator interface in GLU.



Before Tessellation



After Tessellation

Tessellation (cont.)

Reference: Angel page 403

- We can declare a **tessellator object** using GLU

Note: one 'l'

```
GLUtesselator * myTess;
myTess = gluNewTess( );
gluTessBeginPolygon(myTess, NULL);
gluTessBeginContour(myTess);
for (i = 0; i < nvertices; i++)
    gluTessVertex(myTess, vertex[i], vertex[i]);
gluTessEndContour(myTess);
gluTessEndPolygon(myTess);
```

- You also need to register tessellation callbacks:

```
gluTessCallback( GLUtesselator *  t, GLenum which,
GLvoid (*CallBackFunc)() );
```

- Note that Angel contains typos and doesn't mention callbacks!

Other OpenGL functions

- To clear the colour part of the frame buffer:

```
glClearColor(r,g,b,a); /* this only needs to be set
once if you don't change the clear colour */
glClear(GL_COLOR_BUFFER_BIT);
```

Other parts of the buffer (e.g. Depth buffer) can be cleared by a logical or (' | ') with the appropriate constant.

- GLUT coordinates are from the top left (0,0) increasing *x* moves from left to right, increasing *y* goes from top to bottom.
- OpenGL places the origin at the bottom left, increasing *x* moves from left to right, increasing *y* goes from bottom to top.

Matrix Modes

- To transform 2D objects defined in **world coordinates** into rasterized pixels defined in 2D **screen coordinates** OpenGL uses two matrices — the **model-view matrix** and the **projection matrix.**

- These matrices are part of the OpenGL state. To set either of these matrices we preform the following steps:

  - Identify the matrix we want to change

  - Set the matrix to the identity matrix

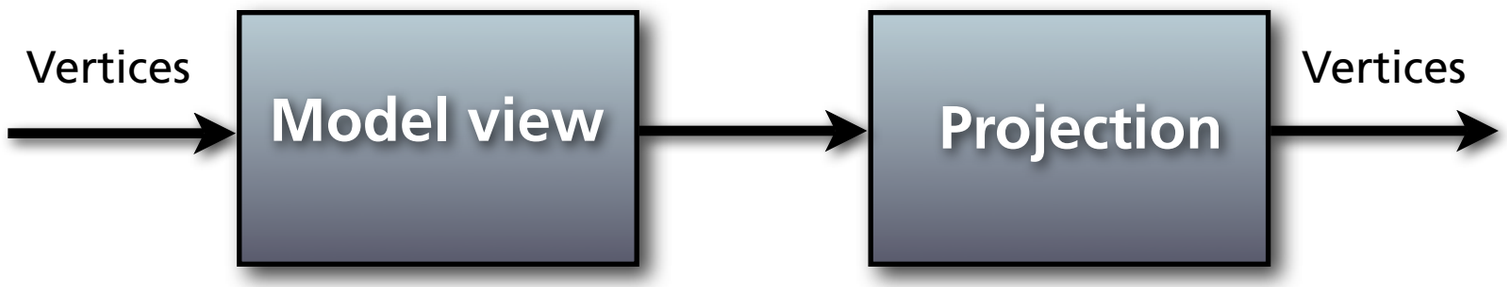
  - Change the identity matrix

- For example to set the two dimensional clipping window:

```
glMatrixMode( GL_PROJECTION );
glLoadIdentity( );
gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
```

Matrix Modes (cont.)

- The model view matrix transforms objects relative to the camera.
- The projection matrix forms the image through projection and helps with clipping by mapping vertices to a normalized coordinate system.
- We can also save and restore the current matrix on a stack using `glPushMatrix( )` and `glPopMatrix( )`



Errors in OpenGL

- You can check for errors in the GL system via

  ```
  GLenum glGetError( );
  ```

  This returns the error type or `GL_NO_ERROR` if no error has been made. Once an error is made, no other errors are recorded until `glGetError` has been called.

- `glGetError` should be called in a loop, until it returns `GL_NO_ERROR` if all error flags are to be reset.

- Most errors do not effect the GL state or frame buffer contents, with the exception of `GL_OUT_OF_MEMORY`.

- The GLU provides a function to return the error number as a string

  ```
  GLubyte * gluErrorString( GLenum error )
  ```

- GLUT provides a function that lets you obtain GLUT state

  ```
  int glutGet(GLenum state)
  ```