Graphics Foundations

Part 2

Colors in computer graphics.

How can color be represented?

The CMYK Model

The RGB Model

80864-2

80864-2

The HSV Model

The YUV Model

Luminance Chrominance Both

The RGB Model is most common in computer graphics.

1-bit per pixel. Can either be one or off.

2-bit per pixel. Can be 4 different colors.
This is called indexed color.

4-bit per pixel. Can be 16 different colors.
The color is still indexed.

8-bit per pixel (1 byte = 1 pixel). Can be 256 different colors.
The color is still indexed.

24-bit per pixel. Can be any color we want (16777216 possible colors). This is called RGB color.

Use
to move.
18:50
MobyGames

PIXEL
PIXEL
PIXEL
PIXEL
PIXEL

32-bit per pixel. Any color plus a transparency level. This is called RGBA color.

PIXEL
PIXEL
PIXEL
PIXEL

HDR. 48-bit per pixel or 96-bit per pixel. RGB channels with 16 or 32 bits per channel.

Colors in OpenGL.

RGB and RGBA colors as 0.0 - 1.0 floating point channels.

RGB

1.0, 0.0, 0.0

1.0, 0.5, 0.0

RGBA

1.0, 0.0, 0.0, 0.7

1.0, 0.5, 0.0, 0.2

Clearing the screen.

```
void glClearColor (GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha);
```

```
glClearColor(0.4f, 0.2f, 0.4f, 1.0f);
```

Sets the clear color of the screen.

```
void glClear (GLbitfield mask);
```

```
glClear(GL_COLOR_BUFFER_BIT);
```

Clears the screen to the set clear color.

Vertex colors

```
void glVertexPointer (GLint size, GLenum type,
GLsizei stride, const GLvoid *pointer);
```

Defines an array of vertex position data.

```
GLfloat triangle[] = {0.0f, 0.5f, -0.5f, -0.5f, 0.5f, -0.5f};

glVertexPointer(2, GL_FLOAT, 0, triangle);
```

```
glEnableClientState(GL_VERTEX_ARRAY);
```

```
void glColorPointer (GLint size, GLenum type,
GLsizei stride, const GLvoid *pointer);
```

Defines an array of vertex color data.

```
GLfloat triangleColors[] = {1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
0.0, 1.0};
glColorPointer(3, GL_FLOAT, 0, triangleColors);
```

```
glEnableClientState(GL_COLOR_ARRAY);
```

```
typedef struct {
    float x;
    float y;

    float r;
    float g;
    float b;
} Vertex2D;
```

Array pointer stride and data structures.

```
Vertex2D triangle[3] = {{0.0, 0.5, 1.0, 0.0, 0.0},
                        {-0.5, -0.5, 0.0, 1.0, 0.0},
                        {0.5, -0.5, 0.0, 0.0, 1.0}};
```

```
Vertex2D triangle[3] = {{0.0, 0.5, 1.0, 0.0, 0.0},
                        {-0.5, -0.5, 0.0, 1.0, 0.0},
                        {0.5, -0.5, 0.0, 0.0, 1.0}};

glVertexPointer(2, GL_FLOAT, sizeof(float) * 5, triangle);
glEnableClientState(GL_VERTEX_ARRAY);

glColorPointer(3, GL_FLOAT, sizeof(float) * 5, &triangle[0].r);
glEnableClientState(GL_COLOR_ARRAY);

glDrawArrays(GL_TRIANGLES, 0, 3);
```

Drawing quads.

```
glDrawArrays (GLenum mode, GLint first,
GLsizei count);
```

Render previously defined arrays.

```
glDrawArrays(GL_QUADS, 0, 3);
```

```
GLfloat quad[] = {-0.5f, 0.5f, -0.5f, -0.5f, 0.5f, -0.5f, 0.5f, 0.5f};

glVertexPointer(2, GL_FLOAT, 0, quad);
glEnableClientState(GL_VERTEX_ARRAY);

glDrawArrays(GL_QUADS, 0, 4);
```

Textures and images.

SDL Surface.

```
SDL_Surface *surface = IMG_Load("myimage.png");
```

```
typedef struct SDL_Surface
{
    int w;          // width of the image
    int h;          // height of the image
    int pitch;      // size of a row in bytes
    void *pixels;   // pointer to pixel data

    // …other data

} SDL_Surface;
```

Textures in OpenGL

```
void glGenTextures (GLsizei n, GLuint *textures);
```

```
GLuint textureID;
glGenTextures(1, &textureID);
```

Generates a new OpenGL texture ID.

```
void glBindTexture (GLenum target, GLuint texture);
```

```
glBindTexture(GL_TEXTURE_2D, textureID);
```

Bind a texture to a texture target.

```
void glTexImage2D (GLenum target, GLint level, GLint
internalformat, GLsizei width, GLsizei height, GLint
border, GLenum format, GLenum type, const GLvoid *pixels);
```

```
glTexImage2D(GL_TEXTURE_2D, 0, 4, surface->w, surface->h, 0,
GL_RGBA, GL_UNSIGNED_BYTE, surface->pixels);
```

Sets the texture data of the specified texture target. Image format must be GL_RGBA for RGBA images or GL_RGB for RGB images.

Some image will load as GL_BGRA or GL_BGR format.

```
void glTexParameteri (GLenum target, GLenum pname,
GLint param);
```

Sets a texture parameter of the specified texture target.
We MUST set the texture filtering parameters before the texture can be used.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

Texture filtering parameters.

GL_LINEAR

GL_NEAREST

Putting it all together.

```
GLuint LoadTexture(const char *image_path) {
    SDL_Surface *surface = IMG_Load(image_path);

    GLuint textureID;
    glGenTextures(1, &textureID);
    glBindTexture(GL_TEXTURE_2D, textureID);

    glTexImage2D(GL_TEXTURE_2D, 0, 4, surface->w, surface->h, 0, GL_RGBA, GL_UNSIGNED_BYTE,
surface->pixels);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    delete surface;

    return textureID;
}
```

Texture coordinates.

```
void glTexCoordPointer (GLint size, GLenum type, GLsizei stride, const GLvoid *pointer);
```

Defines an array of vertex texture coordinate (UV) data.

```
GLfloat quadUVs[] = {0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0};
glTexCoordPointer(2, GL_FLOAT, 0, quadUVs);
```

```
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
```

```
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, textureID);
```

Blending

Blending

Enabling blending

```
glEnable(GL_BLEND);
```

Common blending functions.

Alpha blending

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Additive blending

```
glBlendFunc (GL_SRC_ALPHA, GL_ONE);
```

A simple sprite system.

```
void DrawSprite(GLint texture, float x, float y, float rotation) {
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texture);

    glMatrixMode(GL_MODELVIEW);

    glLoadIdentity();
    glTranslatef(x, y, 0.0);
    glRotatef(rotation, 0.0, 0.0, 1.0);

    GLfloat quad[] = {-0.5f, 0.5f, -0.5f, -0.5f, 0.5f, -0.5f, 0.5f, 0.5f};
    glVertexPointer(2, GL_FLOAT, 0, quad);
    glEnableClientState(GL_VERTEX_ARRAY);

    GLfloat quadUVs[] = {0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0};
    glTexCoordPointer(2, GL_FLOAT, 0, quadUVs);
    glEnableClientState(GL_TEXTURE_COORD_ARRAY);

    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    glDrawArrays(GL_QUADS, 0, 4);
    glDisable(GL_TEXTURE_2D);
}
```

```
DrawSprite(emojiTexture, 0.0, 0.0, 45.0);
```

Keeping time.

```
float lastFrameTicks = 0.0f;
```

During our loop:

```
float ticks = (float)SDL_GetTicks()/1000.0f;
float elapsed = ticks - lastFrameTicks;
lastFrameTicks = ticks;
```

“elapsed” is how many seconds elapsed since last frame. We will use this value to move everything in our game.

Basic time-based animation.

```
emojiAngle += elapsed;
DrawSprite(emojiTexture, 0.0, 0.0, emojiAngle);
```

Advanced transforms.

Matrix stack.

Assignment.

- Create a simple 2D scene using textured and untextured polygons.
- You can use any images you want, but feel free to use the assets in the class github repo.
- At least one element must be animated.
- You must use at least 3 different textures.
- At least one element of the scene must use vertex colors.
- Commit the source to your github repository and email me the link.