OpenGL color

RGBA or indexed color

We primarily use RGBA

- R = red
- G = green
- B = blue
- A = alpha

Values are specified from 0.0 to 1.0

Alpha = transparency!

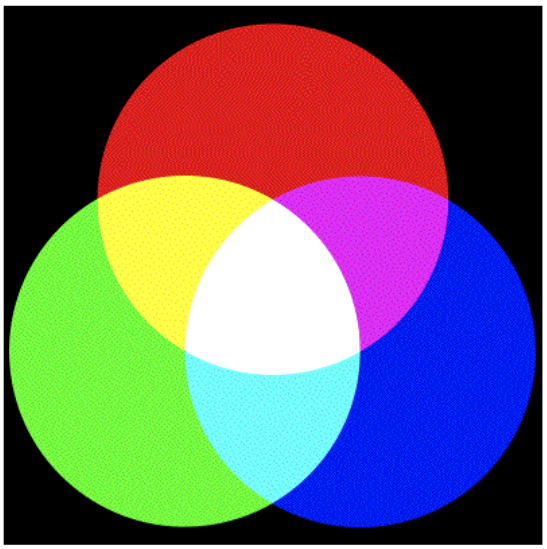

Information Coding / Computer Graphics, ISY, LiTH

Primary colors

Red, Green, Blue

But my Kindergarten Teachers said that it was red, blue and yellow..?

Additive colors!

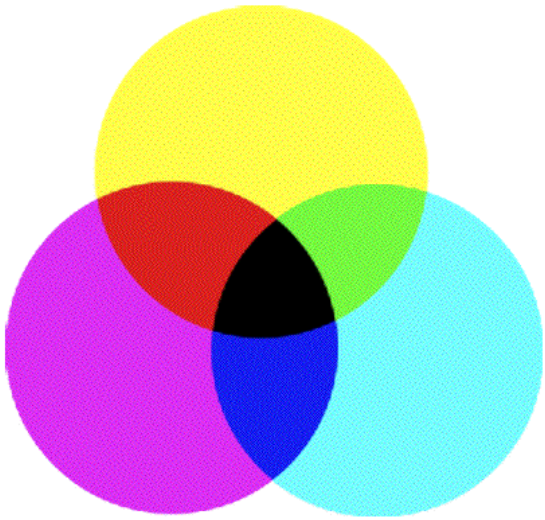


Secondary colors

Magenta, Cyan, Yellow

”Paint”, removes primary colors from white.

Subtractive colors!



Information Coding / Computer Graphics, ISY, LiTH

OpenGL state

The state is a set of internal values in OpenGL.

- Activation of features with glEnable

Any change you make lasts until you change it again!

You don’t automatically get the default back, you get what you used the last time.

Common OpenGL state manipulating calls

Setting state variables:

glPointSize(size)
glLineStipple(repeat, pattern)

Enabling and disabling features:

glEnable(GL_CULL_FACE)
glDisable(GL_TEXTURE_2D)

Selecting objects:

glBindVertexArray, glBindBuffer...

Affine transformations in OpenGL

Rotation, translation, scaling and more. Rotation matrix:

```
GLfloat rotationMatrix[] = {  cos(a), -sin(a), 0.0f, 0.0f,
                              sin(a), cos(a), 0.0f, 0.0f,
                              0.0f, 0.0f, 1.0f, 0.0f,
                              0.0f, 0.0f, 0.0f, 1.0f };
```

Multiply your matrices and upload to the vertex shader!

```
glUniformMatrix4fv(glGetUniformLocation(program, "myMatrix"), 1,
                   GL_FALSE, rotationMatrix);
```

Affine transformations in OpenGL

Doesn't this look slightly more practical?

```
void SetRotationMatrix(GLfloat a, GLfloat *m)
{
   m[0] = cos(a); m[1] = -sin(a); m[2] = 0; m[3] = 0.0;
   m[4] = sin(a); m[5] = cos(a); m[6] = 0; m[7] = 0.0;
   m[8] = 0; m[9] = 0; m[10] = 1.0; m[11] = 0.0;
   m[12] = 0.0; m[13] = 0.0; m[14] = 0.0; m[15] = 1.0;
}
```

You make this any way you see fit. Or you use one out of many third party solutions.

Information Coding / Computer Graphics, ISY, LiTH

Multiple transformations

You can multiply in the vertex shader...

```
in  vec3 inPosition;
uniform mat4 mdlMatrix;
uniform mat4 camMatrix;
uniform mat4 projMatrix;

void main(void)
{
    gl_Position = projMatrix * camMatrix * mdlMatrix * vec4(inPosition, 1.0);
}
```

but global transformations are better done on CPU. (Why?)

Multiple transformation example

Vertex shader still only multiplies one matrix

Uses a simple vector/matrix library

```
Rz(Pi/4, rot);
T(0, -1, 0, trans);
Mult(rot, trans, total);
glUniformMatrix4fv(glGetUniformLocation(program, "myMatrix"),
          1, GL_TRUE, total);
```

Beware that the order of transformations matter!

Information Coding / Computer Graphics, ISY, LiTH

OpenGL evolution:

1.0, 1.1, 1.2: Early versions. Immediate mode.

1.4: Multitexture, shaders are extensions, FBOs are extensions.

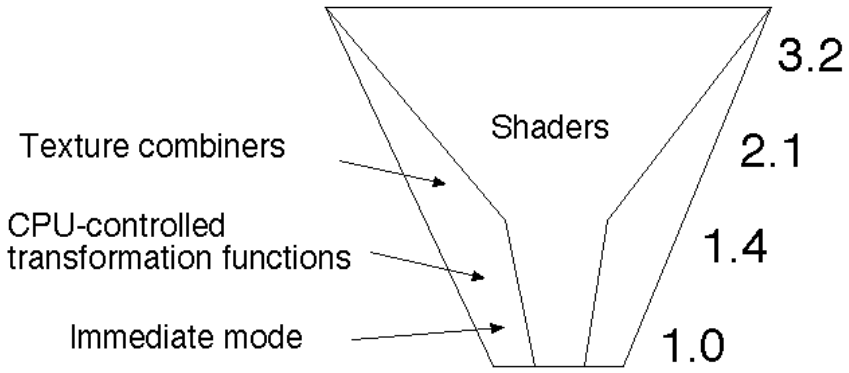
2.1: Shaders and FBOs are integrated.

3.0: Old calls deprecated.

3.2: Old calls are removed (unless you run in compatibility mode).

4.x: Support for new (Fermi) features.

OpenGL is trimming away some fat



...but it isn't disappearing just like that.
"Compatibility extension" for 3.2

Information Coding / Computer Graphics, ISY, LiTH

"How can I teach modern OpenGL without messing the course up for my students?"

Answer: It is harder, a steeper start, but it will eventually pay off. The new way is efficient and gives you better control. Fewer functions and built-in variables to keep track of.

More direct control over transforms makes programming closer to the theory.

We start as easy as possible, package things when you don't want to worry about it.

Goal of this introduction

To provide you with some basic insights in how to use OpenGL to try simple graphics operations *in the simplest way possible*

To see how the graphics operations (transformations) I just presented can appear in working code.

We do not - yet - cover high-performance methods. But we will.

Information Coding / Computer Graphics, ISY, LiTH

Next lecture:

Adding the third dimension!

All this and more. Transformations. Camera placement and projection.