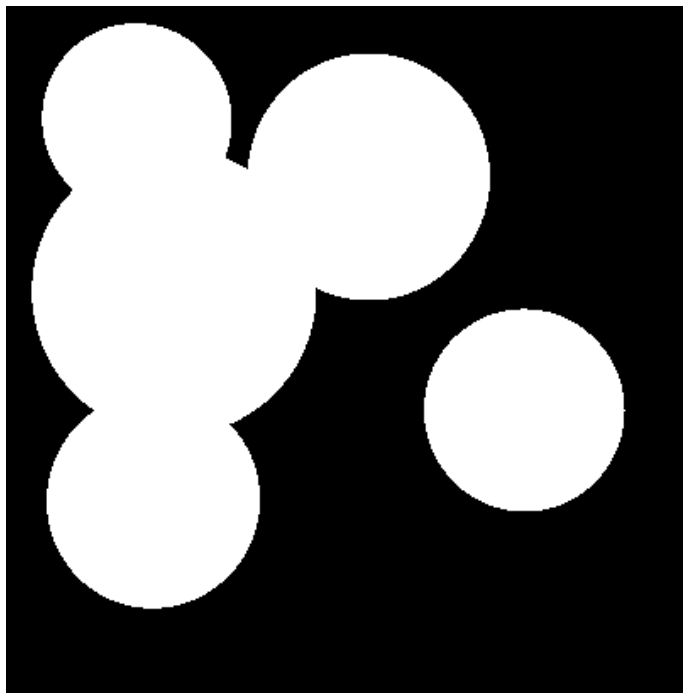
Computer Graphics

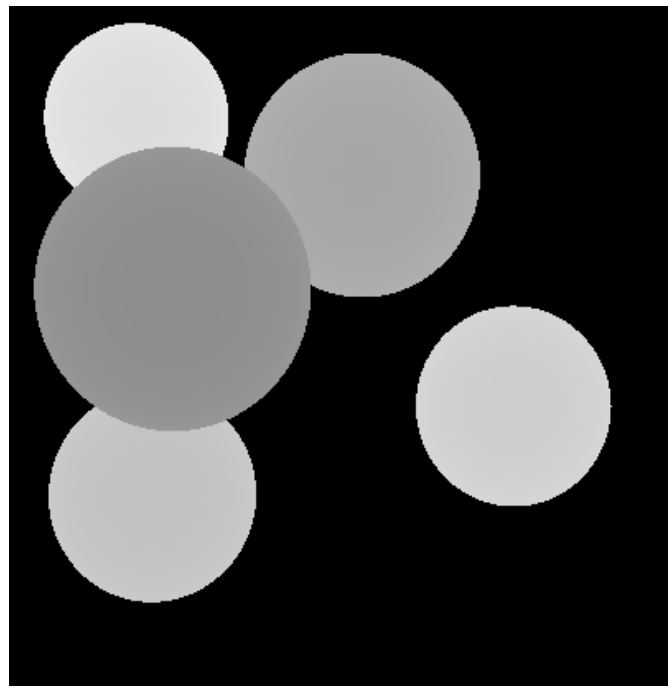
week 2

Jan Paul Posma (1775707)
Roan Kattouw (1770993)

Raytracer: z-buffer display

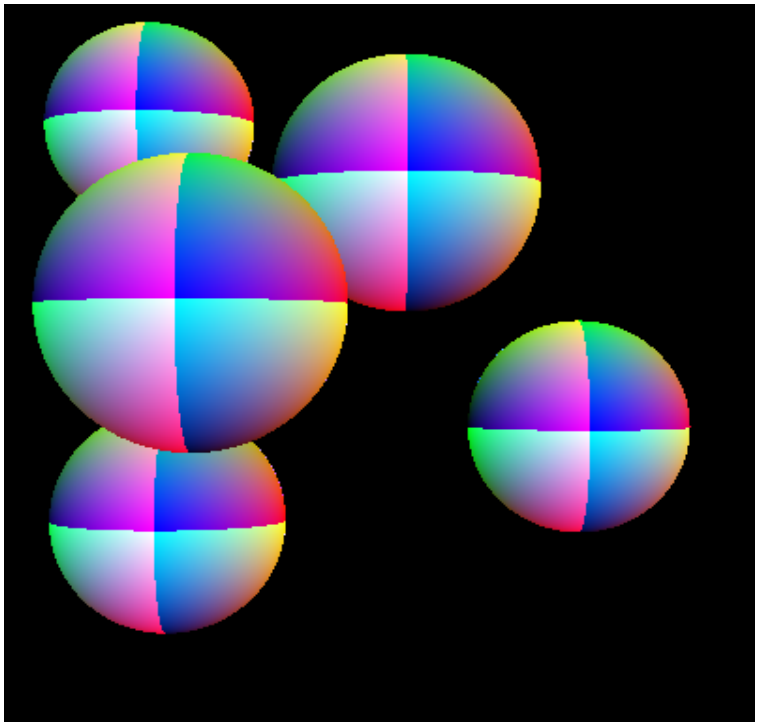


incorrect values

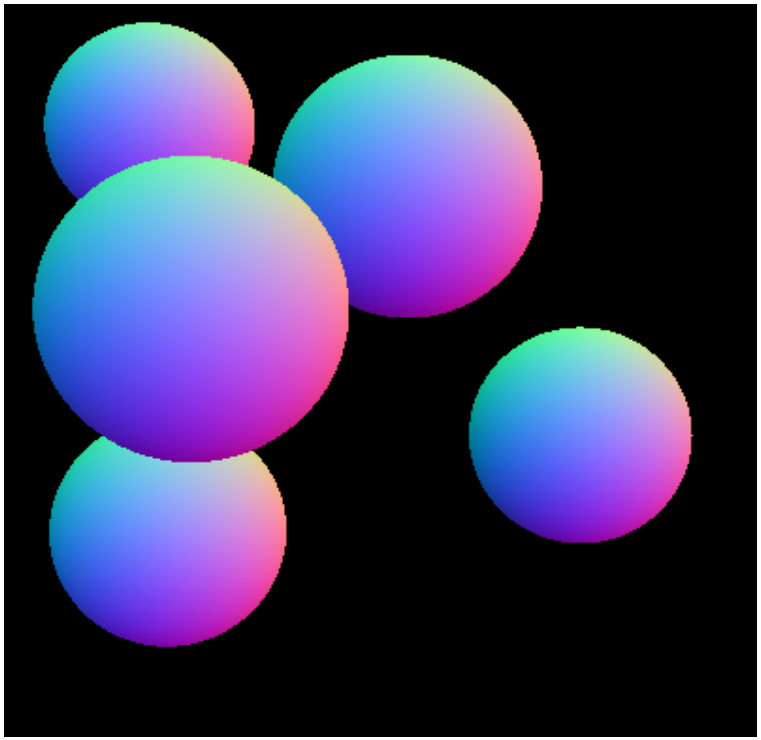


better values

Raytracer: normal display

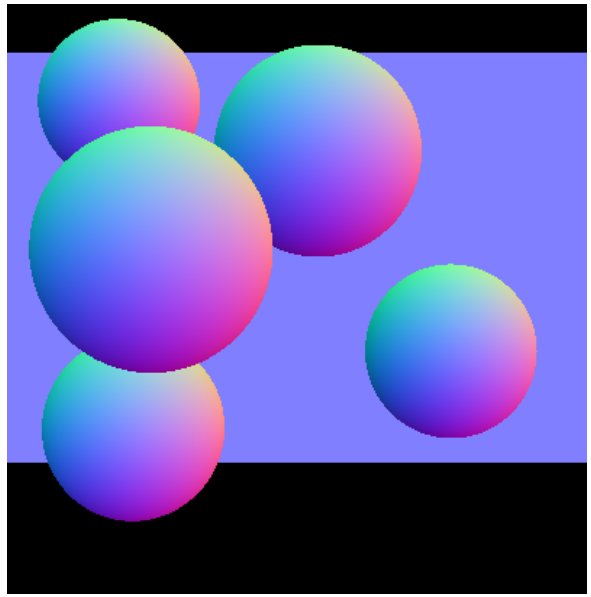


Using normal vectors
as RGB values

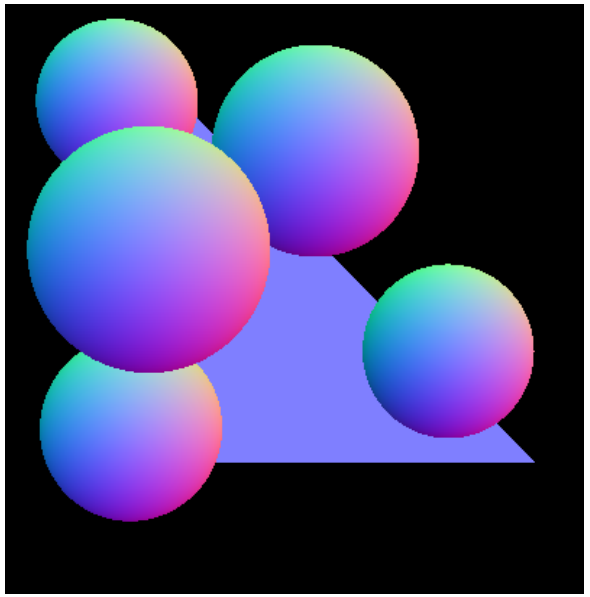


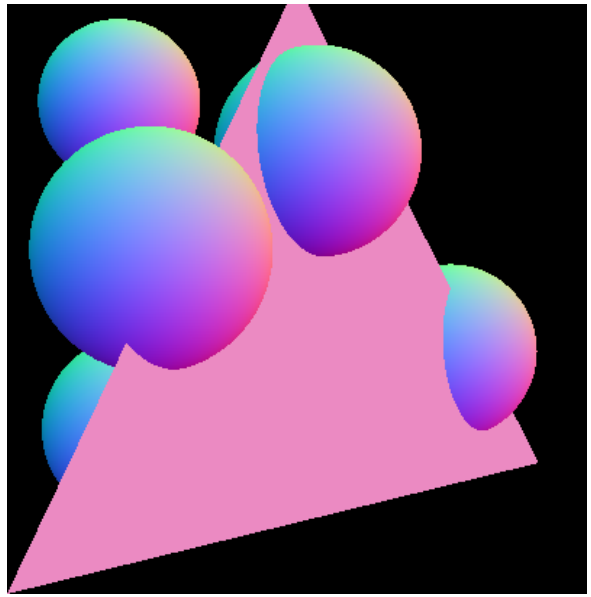
Correctly scaling
values to color space

Raytracer: triangles

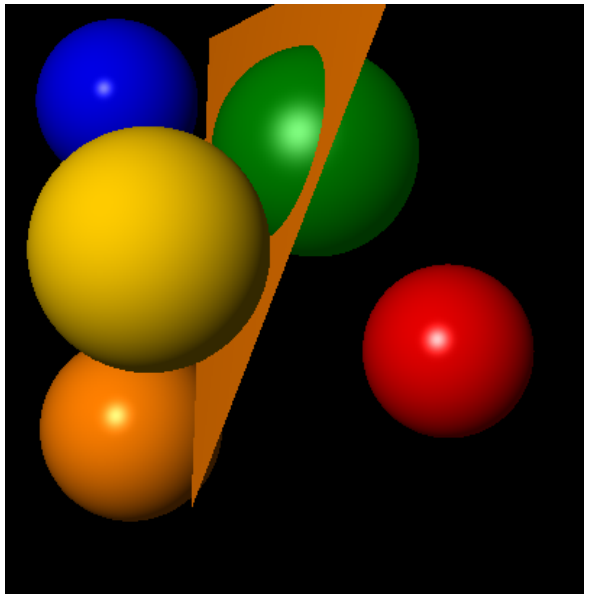


Building the triangle intersection algorithm

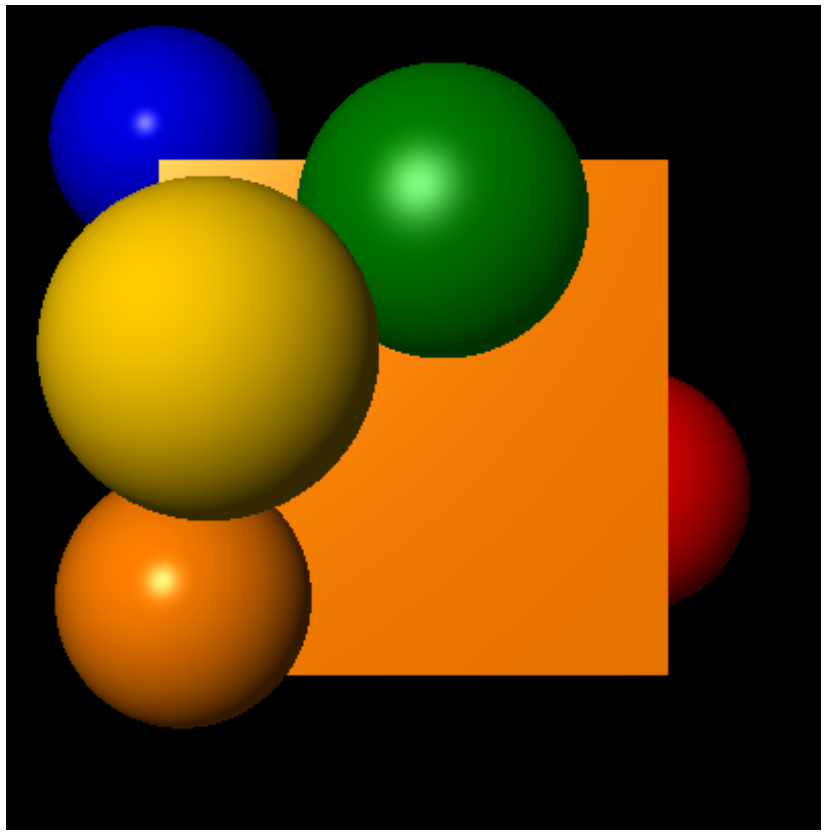




Finished examples



Raytracer: quads



```
Hit Quad::intersect(const Ray &ray)
{
    Hit h1 = Triangle(p1, p2, p3).intersect(ray);
    Hit h2 = Triangle(p1, p3, p4).intersect(ray);
    double inf = std::numeric_limits<double>::infinity();

    if (!(h1.t < inf || h2.t < inf)) return Hit::NO_HIT();

    return ( (h1.t < inf) ? h1 : h2);
}
```

As all points have to be in the same plane, we can simply use two triangles!

Warning: parameters have to be called in the right order! In our case, counter-clockwise with respect to the camera.

Bonus: other shapes

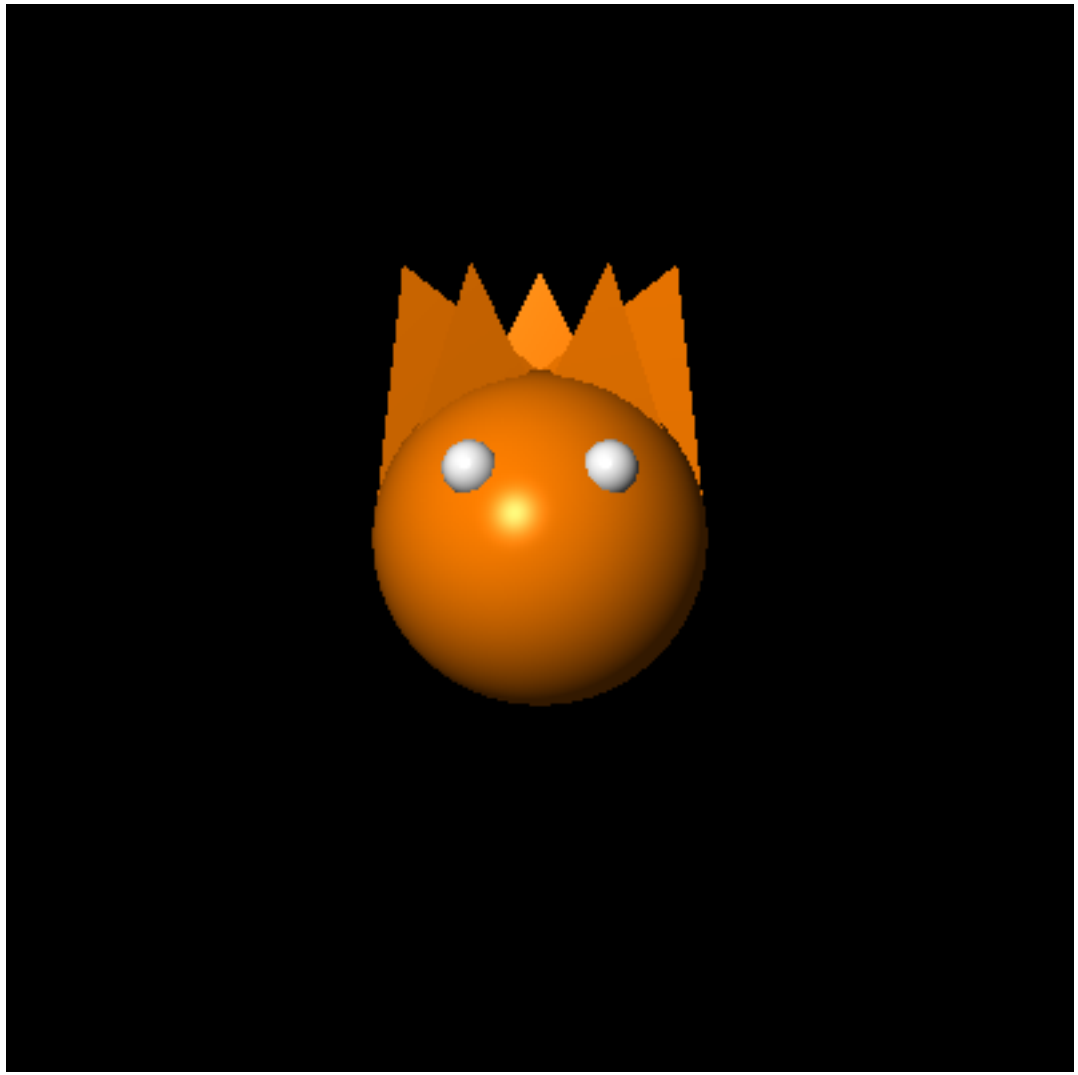
Most other shapes require model-view matrices to place them correctly, perhaps something to implement later.

Explanation of implementing implicit surfaces is not that easy to find, but this is a good place to start: http://www.cl.cam.ac.uk/teaching/1999/AGraphHCI/SMAG/node2.html

Torus: requires solving 4th order polynomial! A torus is a common shape in engineering, and cutter machines actually spend 10% computational time solving these polynomials!
http://en.wikipedia.org/wiki/Quartic_function

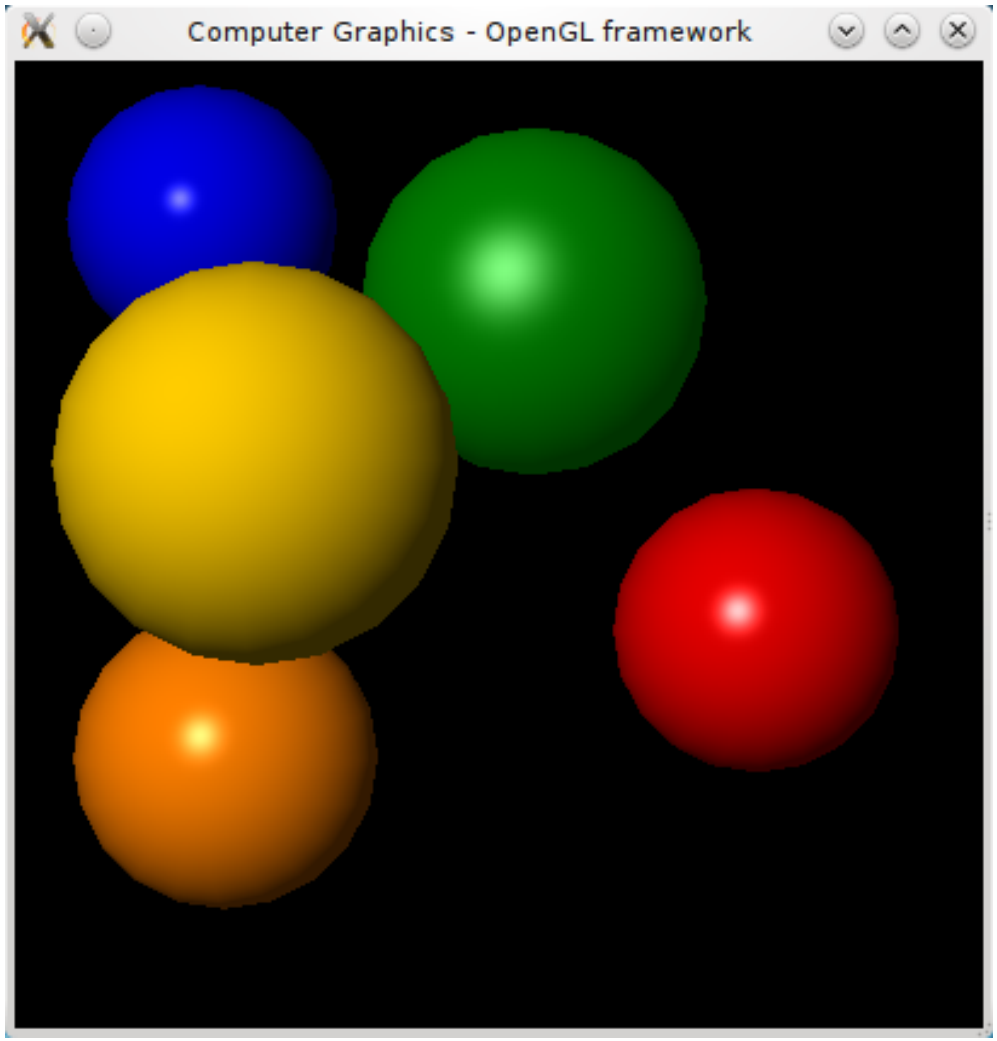
We ended up not implementing these shapes (yet).

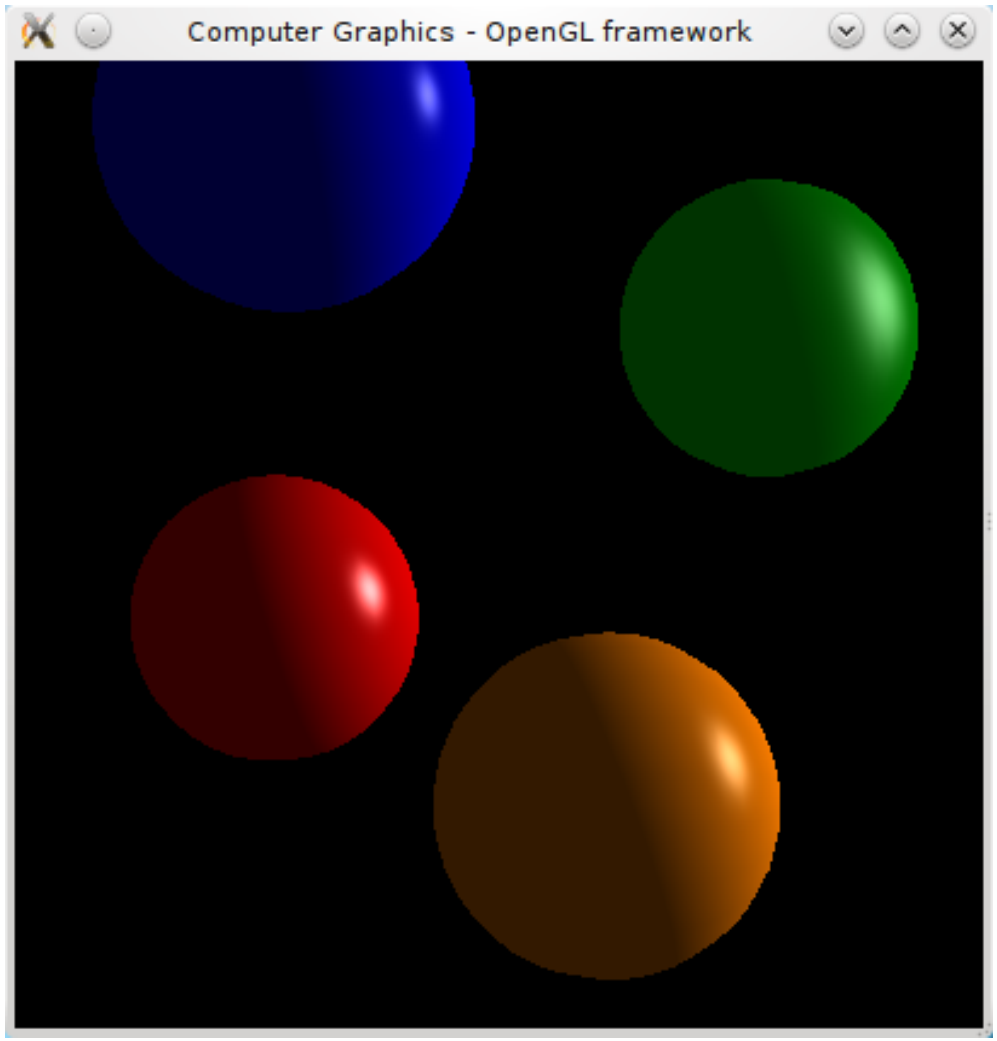
Bonus: other scenes!

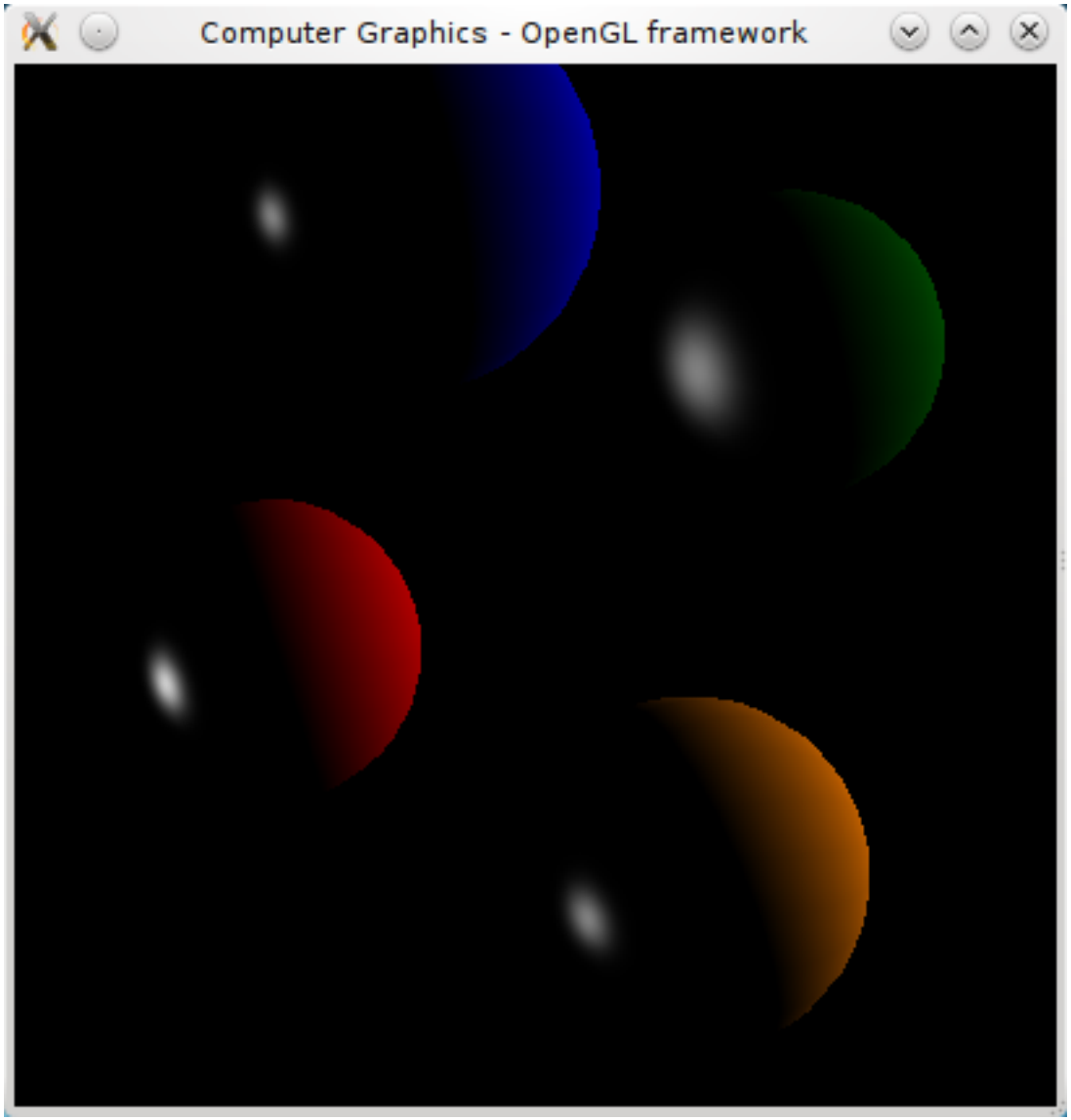


Can't wait to implement anti-aliasing! ;-)

Phong lighting in OpenGL







- Specular reflections on the back side :P
- No ambient light

```
-   V = normalize(vertexPos);
+   V = normalize(-vertexPos);
```