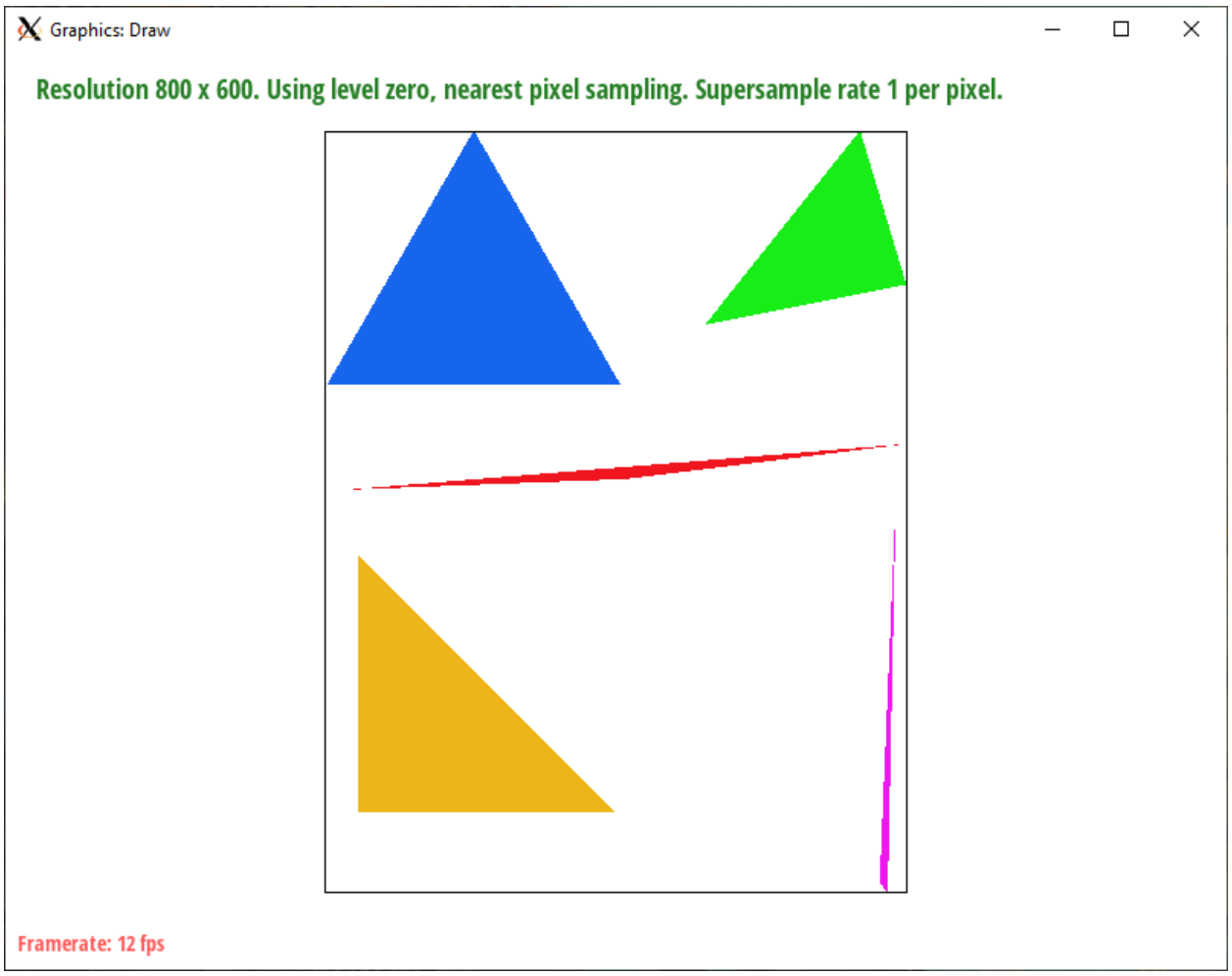
Task 1

I implemented 2 versions of rasterization:

- In the first version, I rasterize the triangle using bounding box. By respectively sampling each and every pixels within the bounding box, I calculated their barycentric coordinate using the triangle vertices. Any pixel with bary coor not within the bound of (0, 1) will be discarded. Others will be rasterized. This method is as efficient as the bounding box method.
- In the second version, I used the classic line sweep method. First, I sorted vertices using y coordinates, and split the triangle into upper half and lower half. For different parts, I will render each points by calculating the intersection bewtween a horiontal line and the two sides of splitted triangles. This method is more efficient. However, it is not friendly to parallelism.

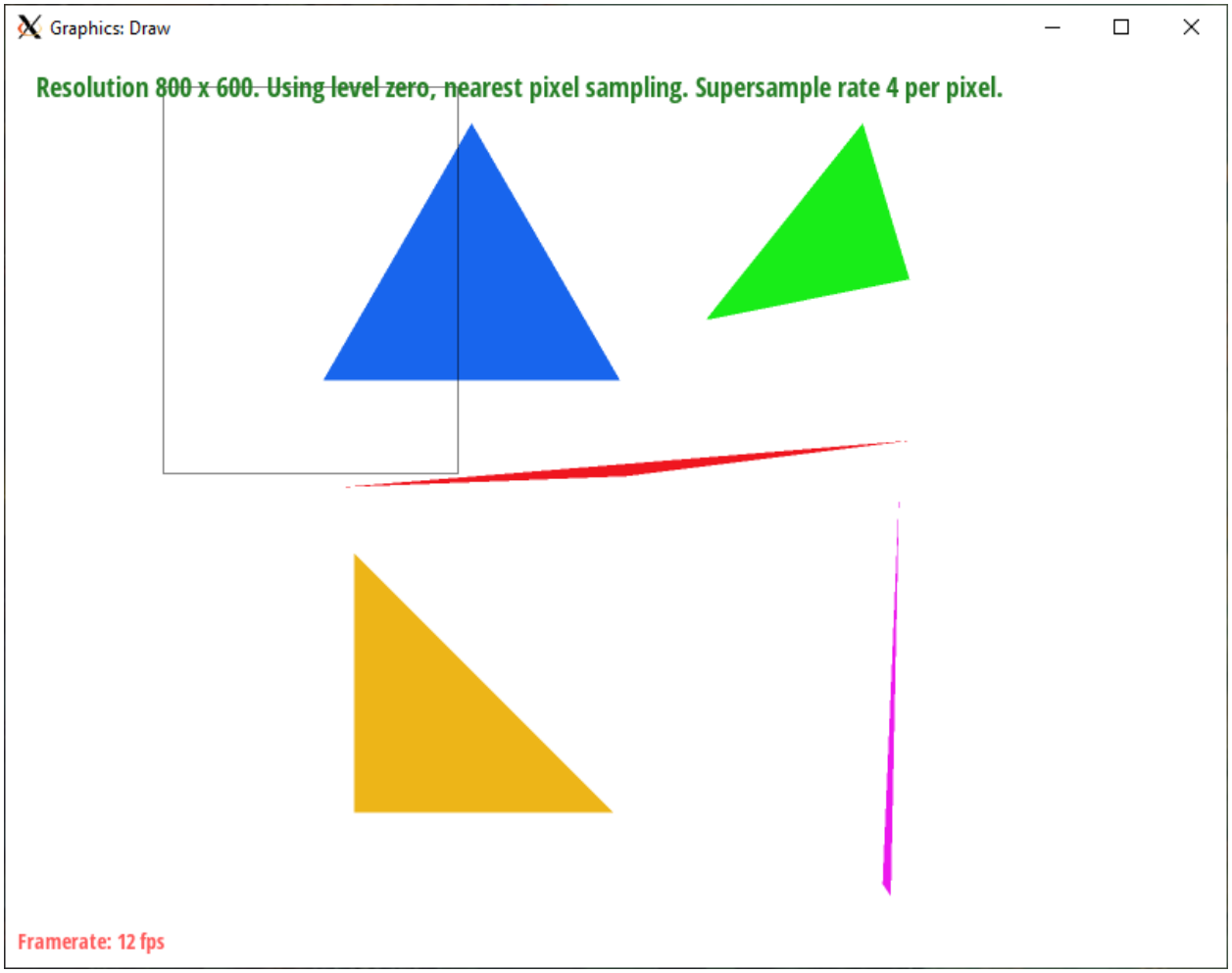


SCREENSHOT

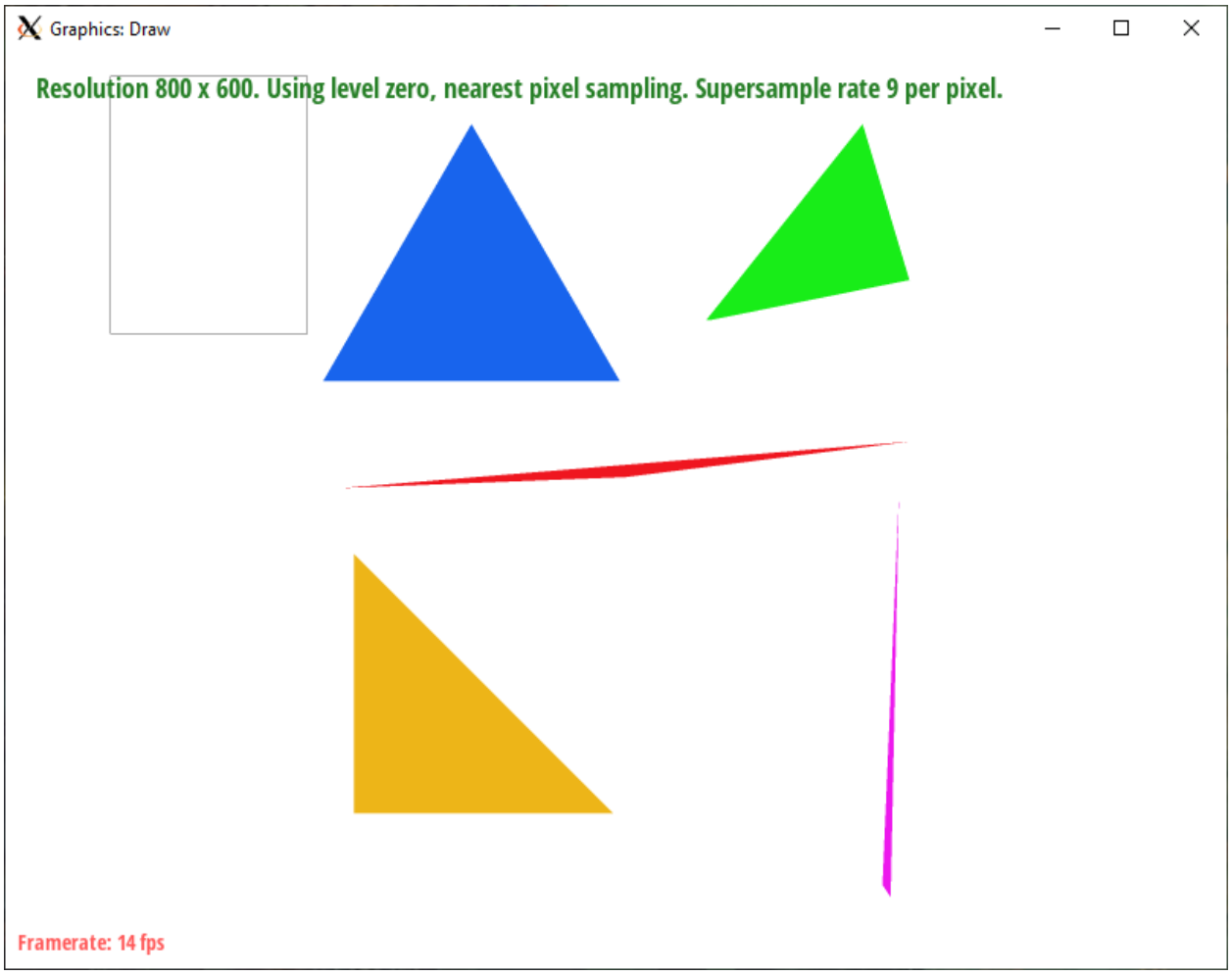
Task2

- Supersampling method basically upsample the original render buffer. For example, render a 1024×1024 frame into a 2048×2048 framebuffer. And then, downsample the larger buffer into the original resolution. In this way we can deal with aliasing in a solid way, as we indeed increase the sampling frequency, with a cost of larger buffer and lower rendering speed.

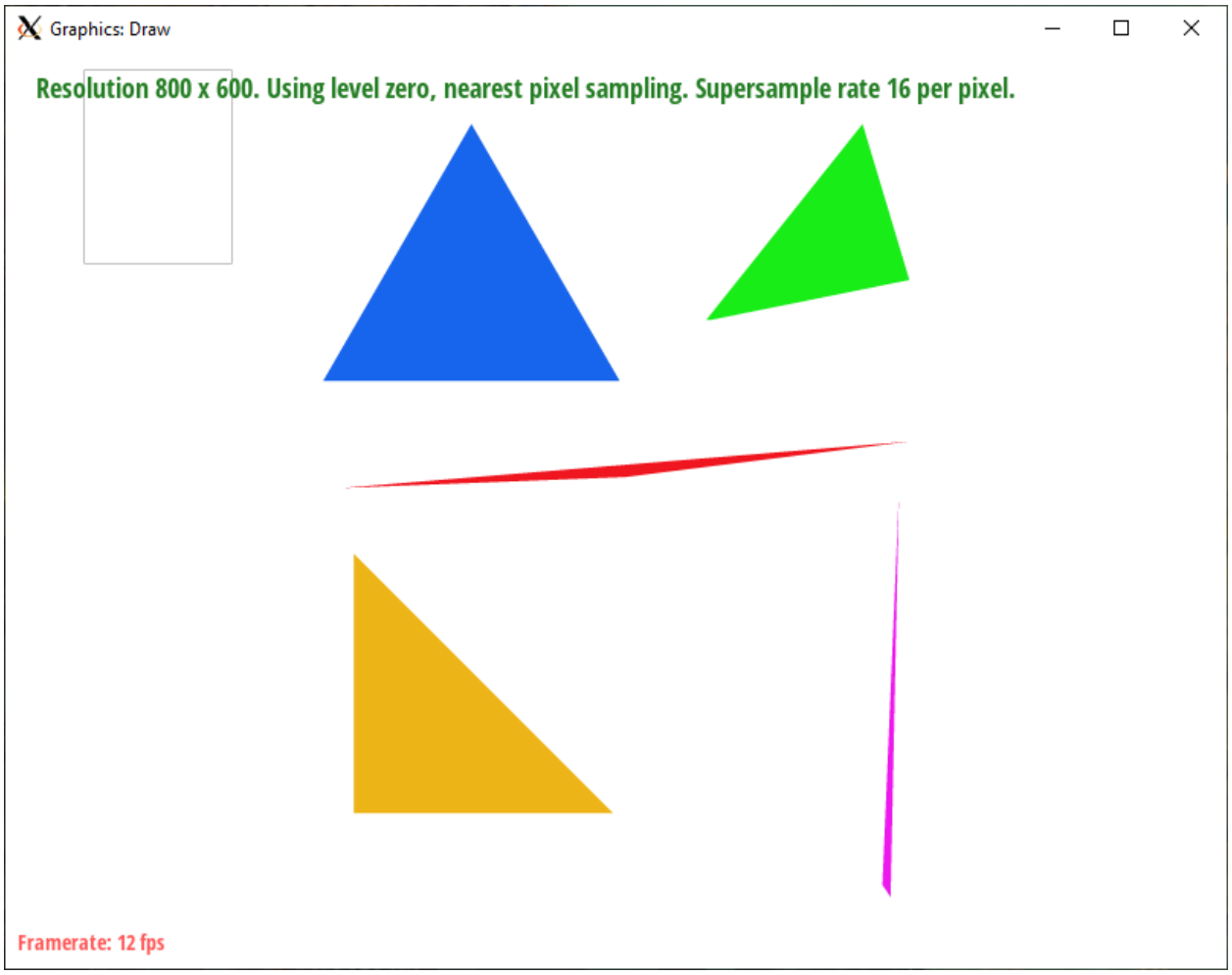
- In my code, I re-allocated the rgb-framebuffer each time user reset sample rate. At the end of rendering pipeline, I will resolve the rgb-framebuffer to the lower resolution buffer.
- ssaa 2x



- ssaa 3x

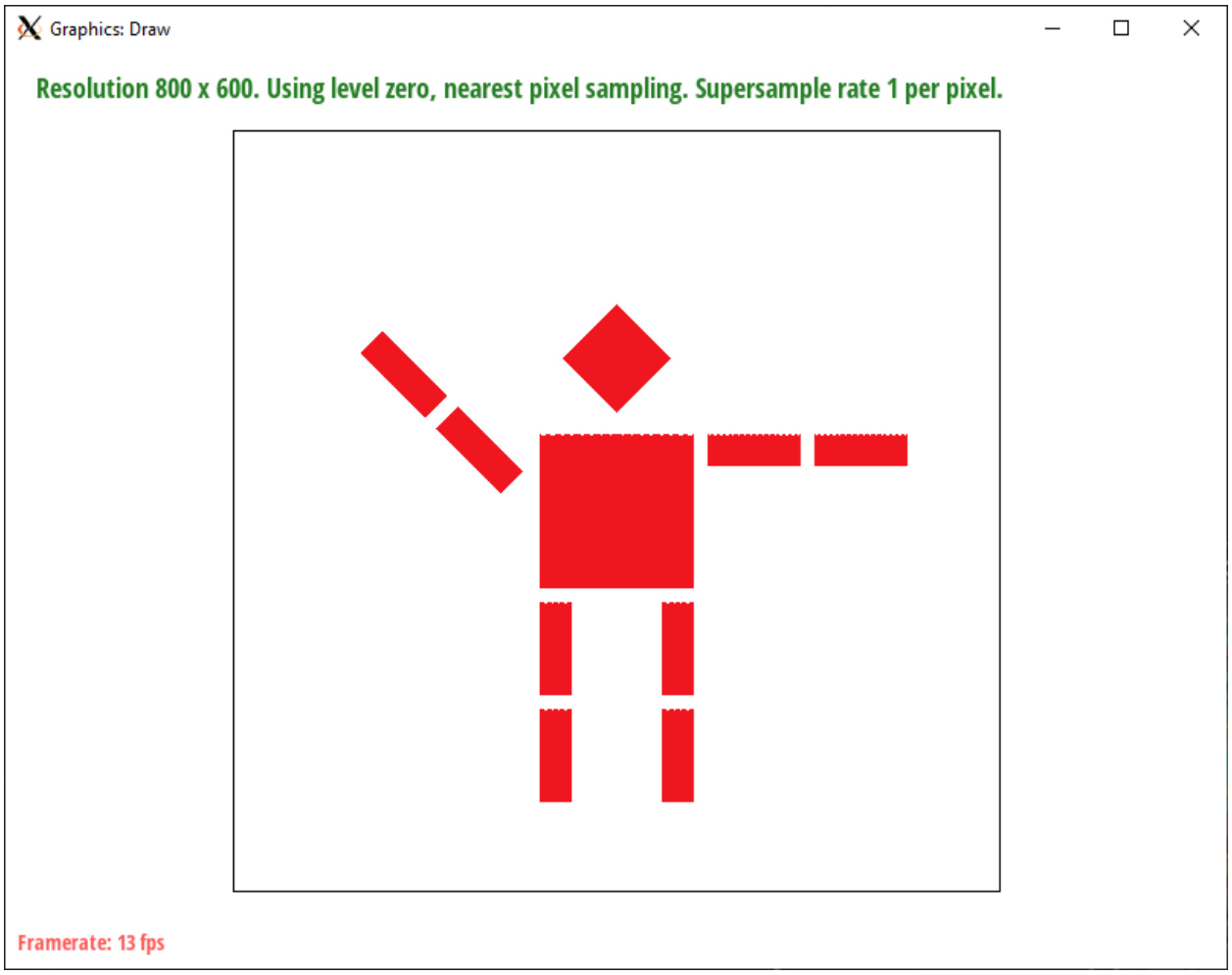

Graphics: Draw
Resolution 800 x 600. Using level zero, nearest pixel sampling. Supersample rate 9 per pixel.
Framerate: 14 fps

- ssaa 4x



Task 3

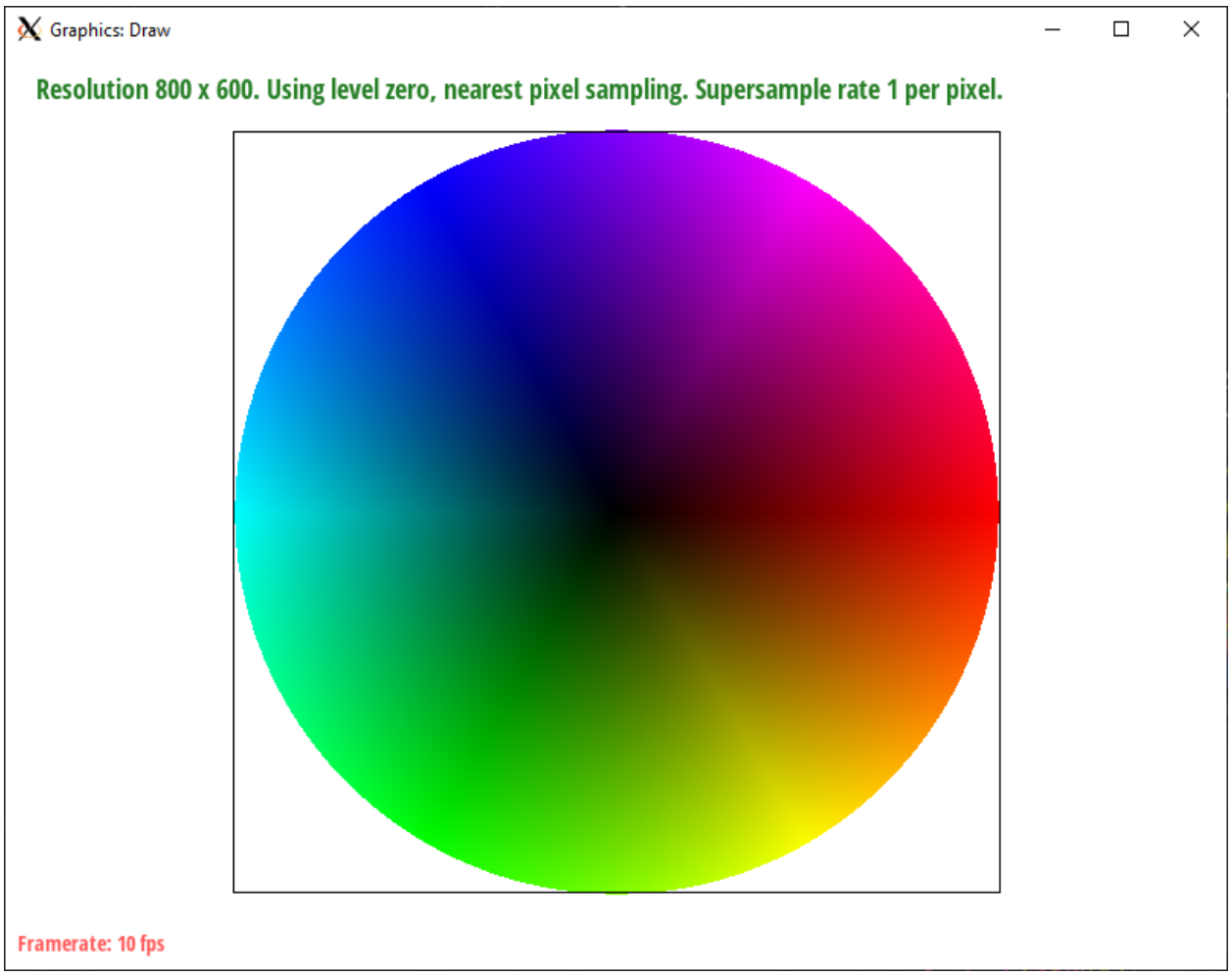
- The cubeman is waving its hand to say hello.



Task 4

- Take a point P inside a triangle ABC. There exists a linear combination of OP by OA, OB and OC. If we represent OP = a $OA + b$ OB + c * OC. Then the coordinate (a, b, c) will be the barycentric

coordinate of P in the triangle ABC.

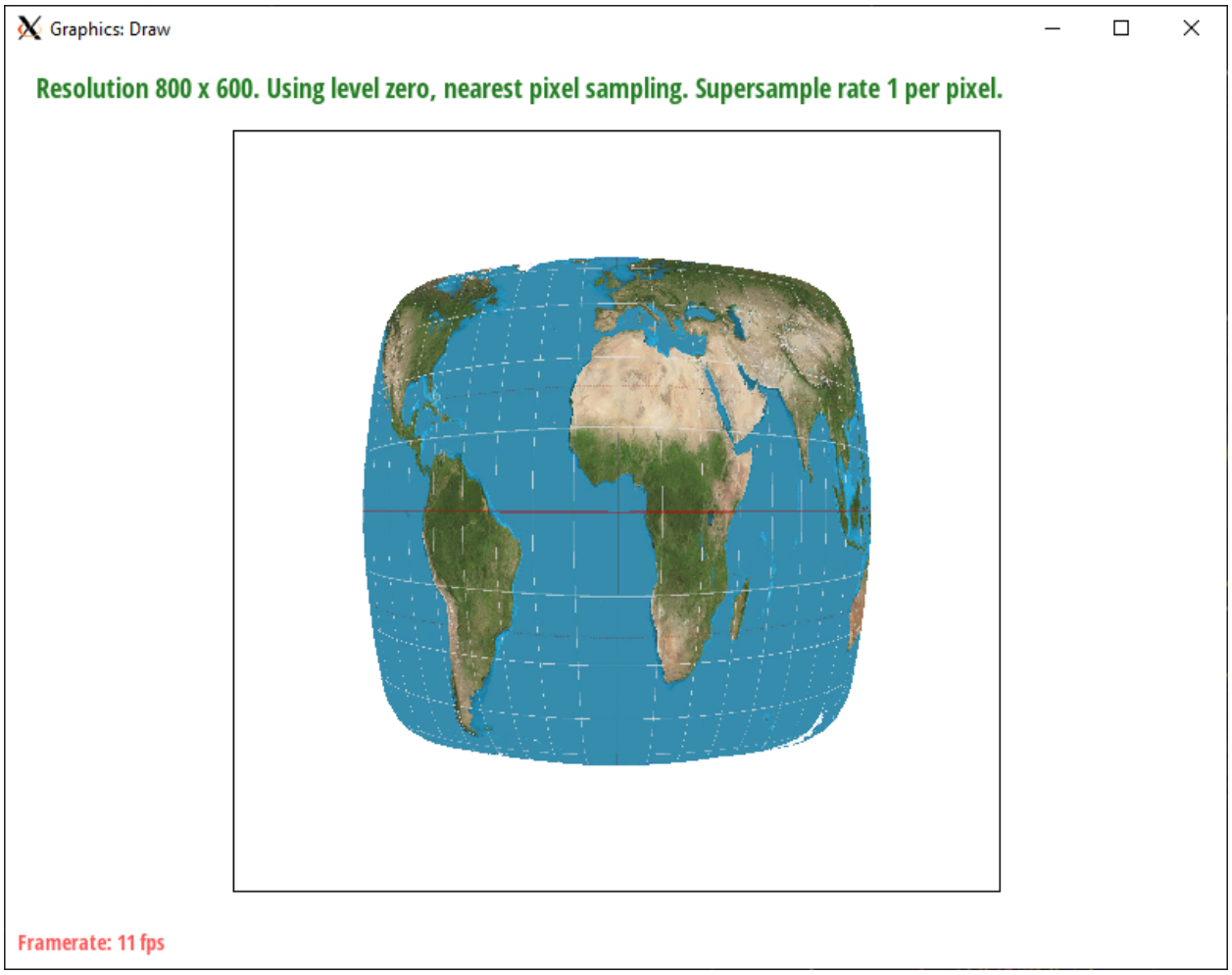


Task 5

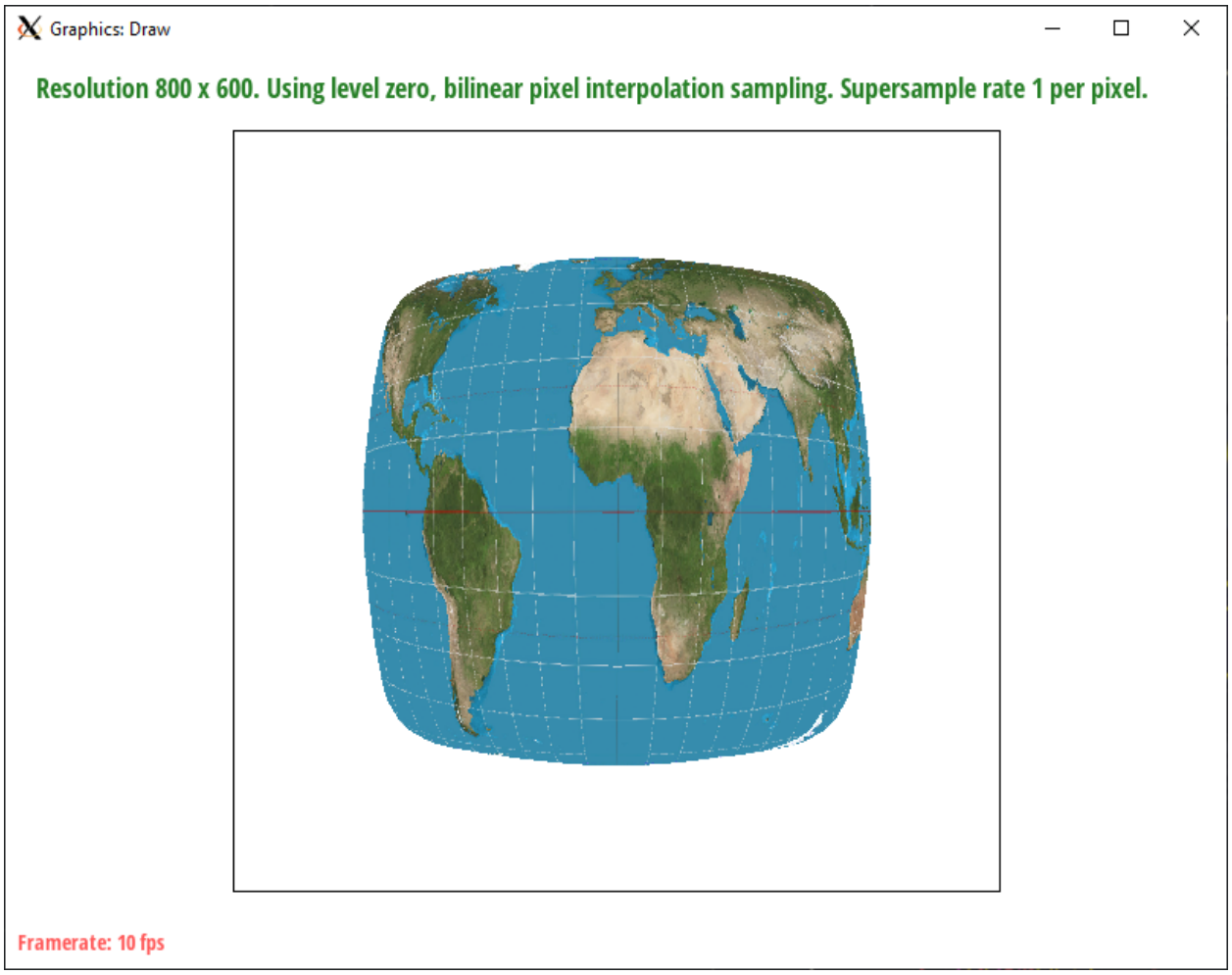
- Pixel texture sampling represents a method where we directly gain the sampled texture of one vertex using its uv coordinate. However, as the resolution of texture is usually limited, the cases are absolutely often where uv coordinates are not mapped to the exact pixel of one texture. This lead to the neccesity of different sampling methods.

- Nearest pixel sampling provides a method where we will simply map a uv coordinates to the nearest pixel. For example, (0.33, 0.33) will be mapped to (1.32, 1.32) at the texture, which is surely not the exact texel. In this case, we will round the coordinates to the nearest integer, which is (1, 1).
- Bilinear pixel sampling means that we will represent a uv coordinates by the linear combinations of its nearest 4 texels. Namely, we will count the weight of its nearest 4 pixels and sum them into the sampled texels.

- Pixel sampling



- Bilinear sampling

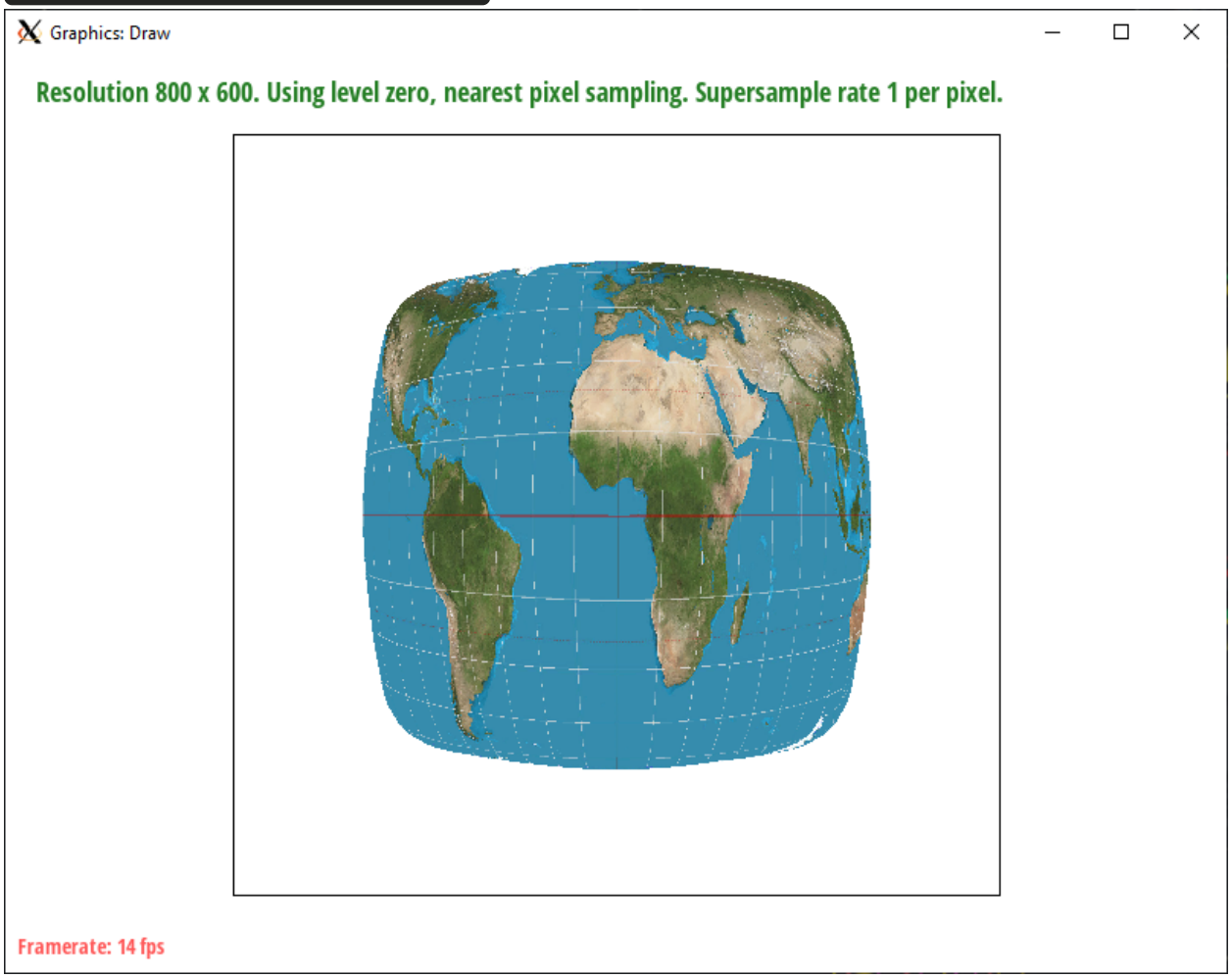


Task 6

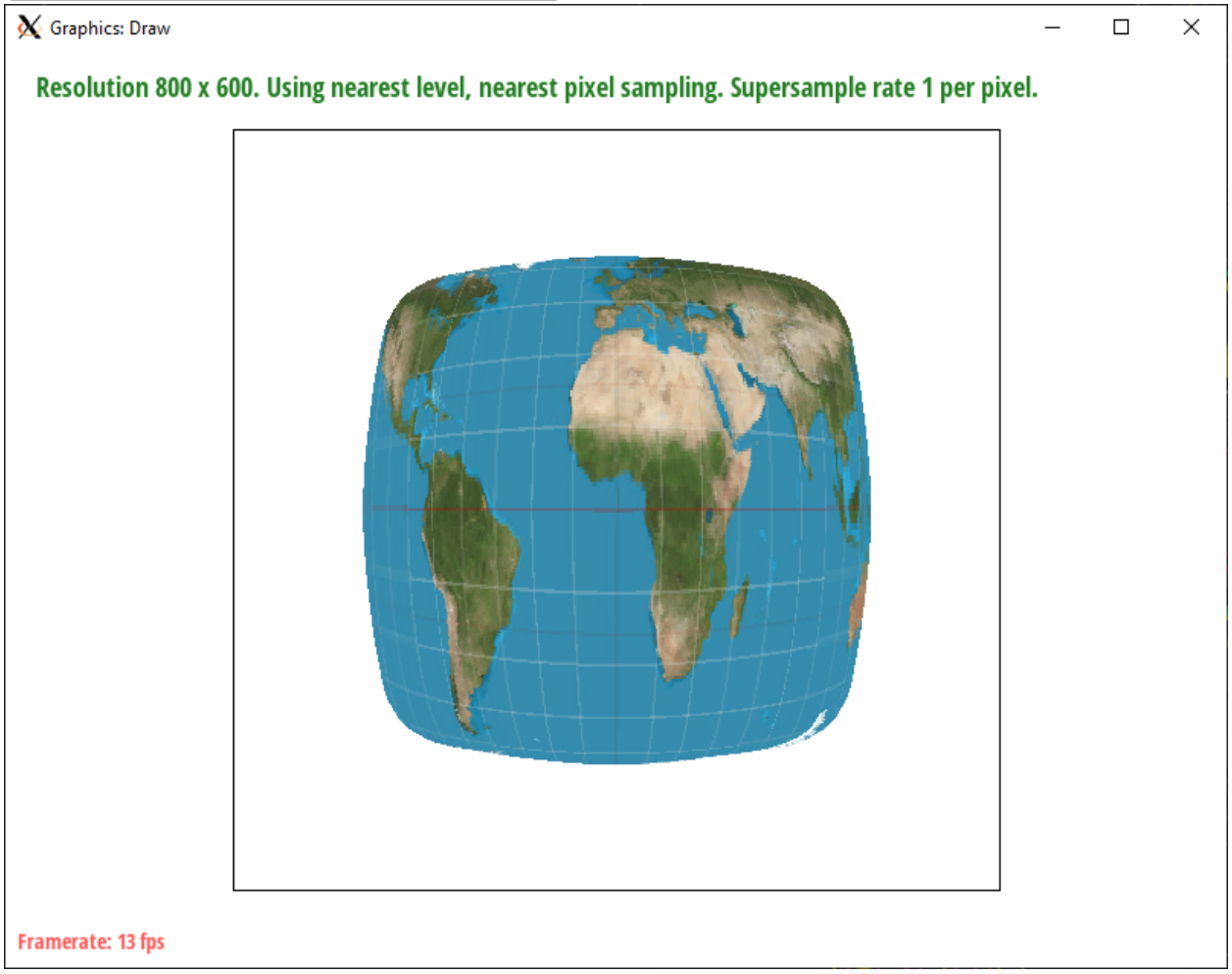
- Level sampling is a method to deal with aliasing that occurs during the texture sampling phase. In level sampling, we will store textures in different resolution, decreasing from original resolution to 1×1 by dividing 2. When sampling texture, we will calculate the comparative scale between texture neighbourhood and vertex neighborbood. The

scale will be used to determine the level of resolution we use to sample the texture.

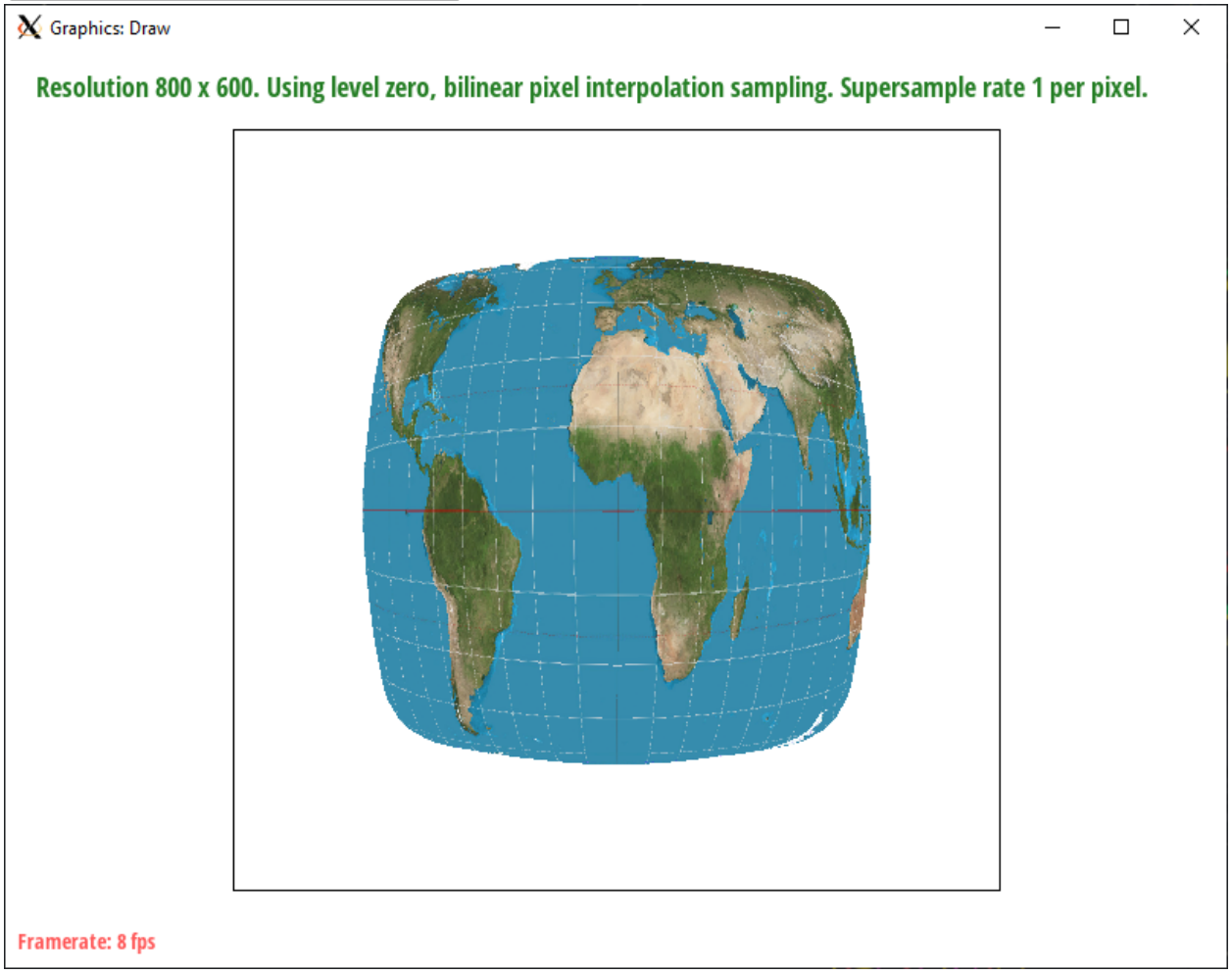
- `P_NEAREST && L_ZERO`



- `P_NEAREST && L_NEAREST`



- `P_LINEAR && L_ZERO`



- `P_LINEAR && L_NEAREST`