INSTANCING

When drawing many **instances** of your model, you'll quickly reach a performance bottleneck because of the many draw calls.
- Example: Drawing 10,000 grass leaves individually to form a lush prairie makes your application very slow.
- Compared to rendering the actual vertices, telling the GPU to render your vertex data with functions like `glDrawArrays` or `glDrawElements` eats up quite a bit of performance since OpenGL must make necessary preparations before it can draw your vertex data (like telling the GPU which buffer to read data from, where to find vertex attributes; all this over the relatively slow CPU to GPU bus).
  - Even though rendering your vertices is super fast, giving your GPU the commands to render them is not.

Instancing is a technique where we draw many (equal mesh data) objects at once with a single render call, saving us all the CPU to GPU communications each time we need to render an object.
- To render using instancing, all we need to do is change the render calls `glDrawArrays` and `glDrawElements` to `glDrawArraysInstanced` and `glDrawElementsInstanced`, respectively.
  - These instanced versions of the normal rendering functions take an extra parameter called the **instance count** that sets the number of instances we want to render.

The instanced functions by themselves are not very useful because you're now just rendering the same object in the exact same position multiple times. Because of this, we will use another GLSL built-in variable in the vertex shader called `gl_InstanceID`.
- When drawing with one of the instanced rendering calls, `gl_InstanceID` is incremented for each instance being rendered, starting from 0.
  - Example: When rendering the 43rd instance of an object, `gl_InstanceID` would be 42 in the vertex shader.

Render 100 evenly-distributed quads in a 10x10 grid across the screen using instanced rendering.
- Instanced Objects

Instanced Arrays

If we need to render an extremely large amount of instances, we will hit a limit on the amount of uniform data we can send to the shaders. One alternative option is known as **instanced arrays**.

Instanced arrays are defined as vertex attributes (allowing us to store much more data) that are updated per instance instead of per vertex.
- With vertex attributes, at the start of each run of the vertex shader, the GPU will retrieve the next set of vertex attributes that belong to the current vertex. However, when defining a vertex attribute as an instanced array, the vertex shader only updates the content of the vertex attribute per instance, which allows us to use the standard vertex attributes for data per vertex and use the instanced array for storing data that is unique per instance.
- Because an instanced array is a vertex attribute, we need to store its content in a vertex buffer object and configure its attribute pointer.

When using instanced arrays, you need to specify `glVertexAttibDivisor(GLuint index, GLuint divisor)` so that OpenGL updates vertex attribute data at the correct time.
- **`index`**: The index of the vertex attribute.
- **`divisor`**: Tells OpenGL when to update the content of the vertex attribute to the next element.
- Example: If `divisor` = 0 (default), OpenGL will update the content of the vertex attribute each iteration of the vertex shader.
- Example: If `divisor` = 1, OpenGL will update the content of the vertex attribute when we start to render a new instance.
  - By setting the divisor to 1, we're effectively telling OpenGL that the vertex attribute at location `index` is an instanced array.
- Example: If `divisor` = 2, OpenGL will update the content of the vertex attribute every 2 instances.

Use an instanced array of the offsets to draw the quads from before. Then, configure the vertex shader to scale the quads depending on which is drawn first (using gl_InstanceID).
- Instanced Arrays

An Asteriod Field

We want to render an asteroid field containing thousands of rocks around a planet.
- Planet model:
- Asteroid model:

To see the performance difference, we're going to first render using `glDrawElements`, and then we're going to implement instanced rendering using `glDrawElementsInstanced`.

Generate an array of model matrices for the asteroids.
- Asteroid Model Matrices

Now, load the planet and rock models (using the Model class and model shaders from the model loading chapter), and draw them without instancing.
- Asteroid Belt without Instancing

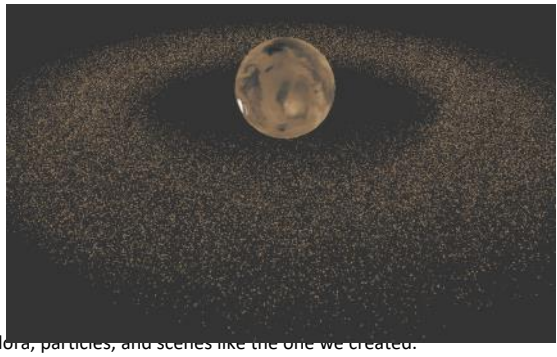
If you try to raise the `amount` of asteroids much higher, you will start to get noticable perform drops.
- **NOTE:** For me, I started seeing performance drops at ~5000 asteroids, but depending on your PC specs, you may start experiencing drops at more or less.
- Currently, the scene contains 1001 rendering calls per frame, with 1000 of those calls being taken up by the rock model rendering.
- This is no good, as you can see in the image to the right. The asteroid field looks incredibly sparse. If we want a large asteroid belt, we need to fix these performance issues by drawing the asteroids using instanced rendering.

Implement instanced rendering for the asteroids in the scene by using an instanced model matrix for the asteroids.
- Asteroid Belt with Instancing

Now, with instanced rendering implemented, you can render significantly more asteroids in the field around the planet!

Implement instanced rendering for the asteroids in the scene by using an instanced model matrix for the asteroids.

- Asteroid Belt with Instancing



Now, with instanced rendering implemented, you can render significantly more asteroids in the field around the planet!

- I encourage you to see how many asteroids you can render now before you start experiencing performance drops now; if everything was implemented correctly, it should be magnitudes more.
  - The image to the right demonstrates 100,000 asteroids being drawn on a radius of `200.0f` with an offset of `50.0f`.
    - One rock model has 576 vertices, so that means the there are over 57 million vertices being drawn in the scene with these settings, and no performance drops.
    - Instead of having to make 100,000 draw calls, we only need to make 1 to draw the same amount of asteroids.

You can surely see the power of instanced rendering now. This makes it a commonly used method for rendering grass, flora, particles, and scenes like the one we created.

- Any scene with many repeating shapes can benefit from instanced rendering.

Quad Vertex Data

Friday, May 6, 2022 12:55 PM

```
float quadVertices[] = {
   // Positions      // Colors
   -0.05f,  0.05f,      1.0f, 0.0f, 0.0f,
    0.05f, -0.05f,      0.0f, 1.0f, 0.0f,
   -0.05f, -0.05f,      0.0f, 0.0f, 1.0f,

   -0.05f,  0.05f,      1.0f, 0.0f, 0.0f,
    0.05f, -0.05f,      0.0f, 1.0f, 0.0f,
    0.05f,  0.05f,      0.0f, 1.0f, 1.0f
};
```

Instanced Objects

Friday, May 6, 2022 1:18 PM

instance.vert

```
#version 330 core

layout (location = 0) in vec2 aPos;
layout (location = 1) in vec3 aColor;

out vec3 color;

uniform vec2 offsets[100];

void main() {
   vec2 offset = offsets[gl_InstanceID];
   gl_Position = vec4(aPos + offset, 0.0, 1.0);
   color = aColor;
}
```

instance.frag

```
#version 330 core

out vec4 FragColor;

in vec3 color;

void main() {
   FragColor = vec4(color, 1.0);
}
```

main()

```
Shader instancedQuadsShader{ "Shaders/instance.vert", "Shaders/instance.frag" };

// Instanced Quad VAO Setup
unsigned int colorQuadVAO, colorQuadVBO;
glGenVertexArrays(1, &colorQuadVAO);
glGenBuffers(1, &colorQuadVBO);

glBindVertexArray(colorQuadVAO);
glBindBuffer(GL_ARRAY_BUFFER, colorQuadVBO);

glBufferData(GL_ARRAY_BUFFER, sizeof(quadVertices), &quadVertices, GL_STATIC_DRAW);

glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 5 * sizeof(float), (void*)0);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 5 * sizeof(float), (void*)(sizeof(float) * 2));

glEnableVertexAttribArray(0);
glEnableVertexAttribArray(1);

glBindVertexArray(0);
glBindBuffer(GL_ARRAY_BUFFER, 0);

// Quad position calculations
glm::vec2 translations[100];
int index = 0;
float offset = 0.1f; // Since objects are centered on their position, we apply an offset to help space them uniformly across the screen
// We're displaying the quads in a 10x10 grid, so the positions are calculated with that in mind
for (int y = -10; y < 10; y += 2) {
   for (int x = -10; x < 10; x += 2) {
      glm::vec2 translation;
      // Normalizes the position to NDC
      translation.x = (float)x / 10.0f + offset;
      translation.y = (float)y / 10.0f + offset;
      translations[index++] = translation;
   }
}
```

Render Loop

```
instancedQuadsShader.use();

// Loops through the translations array to set the offsets array in the vertex shader
for (int i = 0; i < 100; ++i) {
   glUniform2fv(glGetUniformLocation(instancedQuadsShader.id, std::string("offsets[" + std::to_string(i) + "]").c_str()),
       1, glm::value_ptr(translations[i]));
}

glBindVertexArray(colorQuadVAO);
```

```
glDrawArraysInstanced(GL_TRIANGLES, 0, 6, 100); // Drawing 100 (instanced) quads
```

Instanced Arrays

Friday, May 6, 2022 2:07 PM

instance.vert

```
#version 330 core

layout (location = 0) in vec2 aPos;
layout (location = 1) in vec3 aColor;
layout (location = 2) in vec2 aOffset; // Instanced array

out vec3 color;

void main() {
   // Quads grow from bottom-left to top-right
   vec2 pos = aPos * (gl_InstanceID / 100.0);
   gl_Position = vec4(pos + aOffset, 0.0, 1.0);
   color = aColor;
}
```

main()

```
// We need to set up a VBO to store the instanced array content
unsigned int instanceVBO;
glGenBuffers(1, &instanceVBO);
glBindBuffer(GL_ARRAY_BUFFER, instanceVBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec2) * 100, &translations[0], GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);

// Instanced Quad VAO Setup
unsigned int colorQuadVAO, colorQuadVBO;
glGenVertexArrays(1, &colorQuadVAO);
glGenBuffers(1, &colorQuadVBO);

glBindVertexArray(colorQuadVAO);
glBindBuffer(GL_ARRAY_BUFFER, colorQuadVBO);

glBufferData(GL_ARRAY_BUFFER, sizeof(quadVertices), &quadVertices, GL_STATIC_DRAW);

glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 5 * sizeof(float), (void*)0);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 5 * sizeof(float), (void*)(sizeof(float) * 2));

glBindBuffer(GL_ARRAY_BUFFER, instanceVBO);
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), (void*)0);
glBindBuffer(GL_ARRAY_BUFFER, 0);
glVertexAttribDivisor(2, 1); // Tells OpenGL when to update the content of a vertex attribute to the next element.

glEnableVertexAttribArray(0);
glEnableVertexAttribArray(1);
glEnableVertexAttribArray(2);

glBindVertexArray(0);
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

Asteroid Model Matrices

Friday, May 20, 2022 7:11 PM

main()

```
unsigned int amount = 1000;
glm::mat4 *modelMatrices;
modelMatrices = new glm::mat4[amount];
srand(glfwGetTime()); // initialize random seed
float radius = 100.0f;
float offset = 2.5f;
for(unsigned int i = 0; i < amount; i++)
{
    glm::mat4 model = glm::mat4(1.0f);
    // 1. translation: displace along circle with 'radius' in range [-offset, offset]
    float angle = (float)i / (float)amount * 360.0f;
    float displacement = (rand() % (int)(2 * offset * 100)) / 100.0f - offset;
    float x = sin(angle) * radius + displacement;
    displacement = (rand() % (int)(2 * offset * 100)) / 100.0f - offset;
    float y = displacement * 0.4f; // keep height of field smaller compared to width of x and z
    displacement = (rand() % (int)(2 * offset * 100)) / 100.0f - offset;
    float z = cos(angle) * radius + displacement;
    model = glm::translate(model, glm::vec3(x, y, z));

    // 2. scale: scale between 0.05 and 0.25f
    float scale = (rand() % 20) / 100.0f + 0.05;
    model = glm::scale(model, glm::vec3(scale));

    // 3. rotation: add random rotation around a (semi)randomly picked rotation axis vector
    float rotAngle = (rand() % 360);
    model = glm::rotate(model, rotAngle, glm::vec3(0.4f, 0.6f, 0.8f));

    // 4. now add to list of matrices
    modelMatrices[i] = model;
}
```

Don't worry if this looks daunting. All this code does is transform the x and z positions of the asteroid along a circle with a specified radius and randomly displace each asteroid by the offset, with the y displacement being reduced for a more flat asteroid ring. The asteroid is then randomly scaled and rotated and the model matrix is stored in the `modelMatrices` array.

Asteroid Belt without Instancing

Friday, May 20, 2022 7:21 PM

main()

```
Shader modelShader{ "Shaders/model.vert", "Shaders/model.frag" };

Model planetModel{ "Models/planet/planet.obj" };
Model rockModel{ "Models/rock/rock.obj" };
```

Render Loop

```
// Draw planet
modelShader.use();

glm::mat4 model{ 1.0f };
model = glm::scale(model, glm::vec3(15.0f, 15.0f, 15.0f));
glm::mat4 view = camera.GetViewMatrix();
glm::mat4 projection = glm::perspective(camera.fov, (float)WIDTH / (float)HEIGHT, 0.1f, 2000.0f);

glUniformMatrix4fv(glGetUniformLocation(modelShader.id, "model"), 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(glGetUniformLocation(modelShader.id, "view"), 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(glGetUniformLocation(modelShader.id, "projection"), 1, GL_FALSE, glm::value_ptr(projection));

planetModel.Draw(modelShader);

// Draw non-instanced asteroids
for (int i = 0; i < amount; i++) {
   glUniformMatrix4fv(glGetUniformLocation(modelShader.id, "model"), 1, GL_FALSE, glm::value_ptr(modelMatrices[i]));
   rockModel.Draw(modelShader);
}
```

I would recommend moving the starting position of your camera (since it will start in the middle of the planet) and also increasing its speed so it's easier to traverse the scene.

In case you don't have them, here are the normal model vertex and fragment shaders from the model loading chapter:

model.vert

```
#version 330 core

layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;

out vec2 texCoords;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main() {
   gl_Position = projection * view * model * vec4(aPos, 1.0);
   texCoords = aTexCoords;
}
```

model.frag

```
#version 330 core

struct Material {
    sampler2D texture_diffuse1;
};
uniform Material material;

out vec4 FragColor;

in vec2 texCoords;

void main() {
    FragColor = texture(material.texture_diffuse1, texCoords);
}
```

Asteroid Belt with Instancing

Friday, May 20, 2022 7:39 PM

instancedModelMat.vert

```
#version 330 core

layout (location = 0) in vec3 aPos;
layout (location = 2) in vec2 aTexCoords;
layout (location = 3) in mat4 instancedModelMatrix;

out vec2 texCoords;

uniform mat4 projection;
uniform mat4 view;

void main() {
   texCoords = aTexCoords;
   gl_Position = projection * view * instancedModelMatrix * vec4(aPos, 1.0);
}
```

main()

```
Shader instancedRocksShader{ "Shaders/instancedModelMat.vert", "Shaders/model.frag"};

...

// Setting instanced model matrix vertex attribute
unsigned int asteroidModelVBO;
glGenBuffers(1, &asteroidModelVBO);
glBindBuffer(GL_ARRAY_BUFFER, asteroidModelVBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(glm::mat4) * amount, &modelMatrices[0], GL_STATIC_DRAW);

// For each mesh, sets the vertex attribute pointer for the model matrix data, enables the attributes, and
// communicates them as instanced arrays.
for (int i = 0; i < rockModel.meshes.size(); ++i) {
   unsigned int vao = rockModel.meshes[i].vao; // Had to make vao variable public in Mesh class to accomplish this
   glBindVertexArray(vao);

   // A matrix is technically 4 contiguous vec4 vertex attributes, so they take up 4 vertex attribute locations
   glVertexAttribPointer(3, 4, GL_FLOAT, GL_FALSE, sizeof(glm::mat4), (void*)0);
   glVertexAttribPointer(4, 4, GL_FLOAT, GL_FALSE, sizeof(glm::mat4), (void*)(sizeof(glm::vec4)));
   glVertexAttribPointer(5, 4, GL_FLOAT, GL_FALSE, sizeof(glm::mat4), (void*)(2 * sizeof(glm::vec4)));
   glVertexAttribPointer(6, 4, GL_FLOAT, GL_FALSE, sizeof(glm::mat4), (void*)(3 * sizeof(glm::vec4)));

   glEnableVertexAttribArray(3);
   glEnableVertexAttribArray(4);
   glEnableVertexAttribArray(5);
   glEnableVertexAttribArray(6);

   glVertexAttribDivisor(3, 1);
   glVertexAttribDivisor(4, 1);
   glVertexAttribDivisor(5, 1);
   glVertexAttribDivisor(6, 1);

   glBindVertexArray(0);
}
```

Render Loop

```
// Draw planet
modelShader.use();

glm::mat4 model{ 1.0f };
model = glm::scale(model, glm::vec3(15.0f, 15.0f, 15.0f));
glm::mat4 view = camera.GetViewMatrix();
glm::mat4 projection = glm::perspective(camera.fov, (float)WIDTH / (float)HEIGHT, 0.1f, 2000.0f);

glUniformMatrix4fv(glGetUniformLocation(modelShader.id, "model"), 1, GL_FALSE, glm::value_ptr(model));
glUniformMatrix4fv(glGetUniformLocation(modelShader.id, "view"), 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(glGetUniformLocation(modelShader.id, "projection"), 1, GL_FALSE, glm::value_ptr(projection));

planetModel.Draw(modelShader);
```

```
// Draw instanced asteroids
instancedRocksShader.use();

// Need to manually bind the texture since we're not using rockModel's Draw function.
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, rockModel.textures_loaded[0].id); // Also made textures_loaded public

glUniform1i(glGetUniformLocation(instancedRocksShader.id, "material.texture_diffuse1"), 0);
glUniformMatrix4fv(glGetUniformLocation(instancedRocksShader.id, "view"), 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(glGetUniformLocation(instancedRocksShader.id, "projection"), 1, GL_FALSE, glm::value_ptr(projection));

for (int i = 0; i < rockModel.meshes.size(); ++i) {
   glBindVertexArray(rockModel.meshes[i].vao);
   glDrawElementsInstanced(GL_TRIANGLES, rockModel.meshes[i].indices.size(), GL_UNSIGNED_INT, 0, amount);
}
```