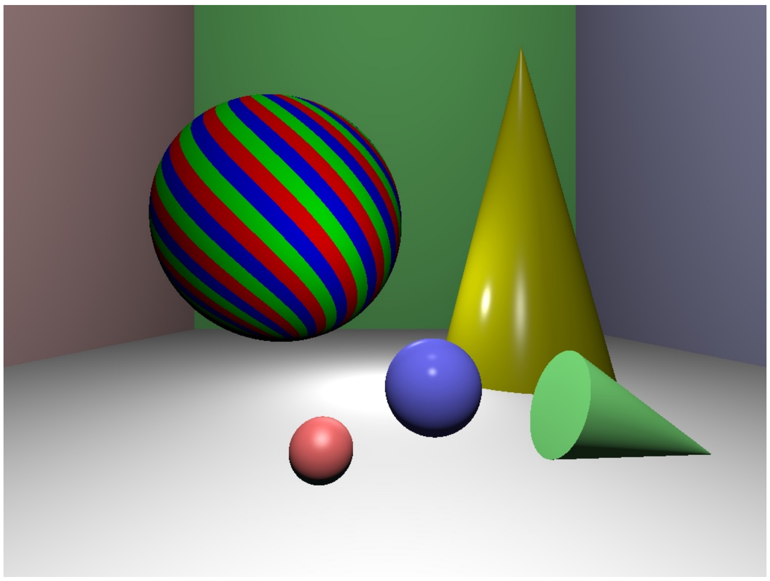
Università della Svizzera italiana

Faculty of Informatics

Computer Graphics (Fall 2021)

Assignment 5: Cone and transformations

October 20, 2021



In this assignment, you will add a new primitive to our raytracer, a cone, and use transformation matrices to position it in the scene. The ray-cone intersection should be performed in the local coordinate space of the cone; therefore, you can limit yourself to implementing the ray-cone intersection for a very simple cone. The transformation matrices will define the final size of the cone and its orientation.

Updated framework

Changes to the framework:

- The definition of the *Object* class contains now variables for three matrices:
  - `transformationMatrix` representing the transformation of the object in the global coordinate system,
  - `inverseTransformationMatrix`, the inverse of this transformation,
  - and `normalMatrix` for the normal vectors transformation in the global coordinate system;
- The *Object* class now also contains the template of the function `setTransformation` which should set the `transformationMatrix` and compute the other two matrices based on it;
- New header file, *"glm/gtx/transform.hpp"*, of the GLM library is included for computing the transformation matrices;
- An empty definition of the *Cone* class.

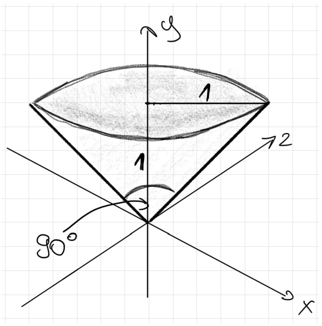
The code is commented. Before continuing with the assignment, please familiarize yourself with the updates and check the code for comments indicating the places which should be modified. As usual, the current template is a solution for the previous codding assignment. Whether you want to start directly with the provided template or continue from your solution to the previous assignment, it is up to you. In the latter case, you are free to copy the above updates to the framework.

Useful functionality of the GLM library

- `glm::mat4` type for 4×4 matrices;
- functions `glm::inverse` and `glm::transpose` allow you to easily compute the inverse and transpose of a given matrix,
- functions `glm::translate`, `glm::scale`, and `glm::rotate` enable an easy definition of translation, scaling, and rotation matrices;
- operation on matrices, vectors can be written as you would normally do on a paper, e.g., you can directly multiply matrix by matrix or matrix by vector, as long as the dimensions agree;
- you can easily extend a `vec3` by a homogeneous coordinate, multiply it by 4×4 and cast it back to `vec3`, e.g., :
  `glm::vec3 newOrigin = matrix * glm::vec4(origin, 1.0);` or
  `glm::vec3 newDirection = matrix * glm::vec4(direction, 0.0);`

Exercise 1 [10 points]

Start by implementing the *Cone* class, such that it represents a simple cone as shown in the image on the right. The cone is **open**, it has 90 degrees opening angle, and the height equal 1. More formally, the equation of the cone is given by $x^2 + z^2 = y^2$ for $y \in [0, 1]$.



To be able to transform the cone you should:

- finish the implementation of the function `Object::setTransformation`,
- implement the intersection with the cone in the local coordinate system of the cone, i.e., before performing the intersection, transform the ray to the local coordinate system using transformation matrices, and then perform the intersection with the transformed ray. Afterwards, transform all the information to the global coordinate system as discussed during the lecture. Please note that you do not have to take care in this exercise about the uv-coordinates. But the intersection point, normal vector, and the distance to the intersection have to be correctly defined in the final *Hit* structure.

Finally, add two cones to the scene. To this end modify the `sceneDefinition` function. Remember to set the transformation matrix for each cone using the function `setTransformation` before adding each cone to the collection of the objects.
You should add two cones as on the image above. The yellow cone should be highly specular. Its height is 12, and the radius of the bottom part is 3. The tip of the cone is at point $(5, 9, 14)$. The green diffuse cone has height 3 and radius at the base 1. The tip is located at point $(6, -3, 7)$, and it is rotated around the z-axis such that it lies with the side wall parallel to the ground. While you do not have to be very accurate to match the material appearance from the image, please make the yellow cone highly specular such that the verification of the normals and lighting computation is easier.

Exercise 2 [5 points]

Extend the implementation of the ray-cone intersection such that the cone is closed; i.e., add a disk that closes the cone. The image above visualizes the closed green cone. You can use the *Plane* class for creating the disk.

Bonus exercise 3 [5 points]

Change the *Sphere* class such that the ray-sphere intersection is performed in the local coordinate system, similarly as for the cone. The intersection routine should consider a simple sphere with radius one and the center at the center of the local coordinate system of the sphere. The position and the size of each sphere should be defined using transformation matrices. Make sure that the texture coordinates are defined in the local coordinate system. Use the new version of the *Sphere* class for all the spheres in the scene. After you achieve this, create an animation in which the textured sphere rotates around its axis and moves up and down at the same time. If you correctly define the texture coordinates, the texture of the sphere will rotate with it. For tips on how to create an animation, please check assignment 2.

Submission

You should submit one ZIP-file via iCorsi containing:

- readme file with information which exercises you solved, the authors of the solutions, and explanation of encountered problems, if any,
- one modified *main.cpp*,
- the final image generated using your code,
- for the Exercise 3, include additionally a 2-second video clip.

Solutions must be returned on October 27, 2021 via iCorsi3