UNIVERSITÄT DES SAARLANDES
Prof. Dr.-Ing. Philipp Slusallek
Computer Graphics Group
Stefan Lemme (lemme@cg.uni-saarland.de)

19. November 2018

Introduction to Computer Graphics
Assignment 4

Submission deadline for the exercises: 26. November 2018

The paper copies for the theoretical parts of the assignments will be collected at the beginning of the lecture on the due date. The programming parts must instead be marked as release before the beginning of the lecture on the due date.
The code submitted for the programming part of the assignments is required to reproduce the provided reference images. The submission ought to tag the respective commit in your group's git repository as well as attach the mandatory generated images to the release. The submission should also contain a creative image show-casing all extra-credit features that have been implemented.
The projects are expected to compile and work out of the box. A successful build by Drone CI is a good indicator. If it fails to do so on our Windows or Linux computers, we will try to do so on the CIP-pool students' lab as a "fallback".
To pass the course you need for every assignment at least 50% of the points.

4.1 Affine Spaces (4 + 4 Points)

Prove that the set of points $A = \{(x, y, z, w) \in R^4 \mid w = 1\}$ is an affine space. What is the associated vector space? You do *not* have to show that the associated vector space is a vector space. What is the difference between a point and a vector in that affine space?

Alternative definition of an affine space: An affine space consists of a set of points P, an associated vector space V and an operation $+ \in V \times P \to P$ that fulfills the following axioms:

(1) `for` $p \in P$ `and` $v, w \in V : w + (v + p) = (w + v) + p$

(2) `for` $p, q \in P$ `there exists a unique` $v \in V$ `such that:` $v + p = q$

4.2 Rotations (10 Points)

Show that an arbitrary rotation around the origin in 2D can be represented by a combination of a shearing in y, a scaling in x and y and a shearing in x in this order. You have to derive the shearing and scaling matrices to an arbitrary rotation T.

$$T = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix}$$

4.3 Homogeneous Coordinates (4 + 4 Points)

a) Show that multiplying the homogeneous point $(x, y, z, w \neq 0)$ with an arbitrary scalar $\alpha \neq 0$ yields an equivalent homogeneous point again.

b) Show that the component wise addition of three homogeneous points $(a_0, b_0, c_0, 1)$, $(a_1, b_1, c_1, 1)$, and $(a_2, b_2, c_2, 1)$ yields the center between that points.

4.4 Homogeneous Lines in 2D (9 Points)

Prove that the cross product between two homogeneous points $p = (p_x, p_y, p_w)$ and $q = (q_x, q_y, q_w)$ yields the homogeneous coordinates of the connecting line.

4.5 Float4 (5 Points)

When doing transformations it is convenient to represent points and vectors uniformly in a `Float4` object. Your task is to implement the `Float4` class. All the arithmetic binary operators should execute component-wise. Points (x, y, z) should transform to $(x, y, z, 1)$, while Vectors (x, y, z) should translate to $(x, y, z, 0)$.
In addition define a 4-component `dot` product on two `Float4` objects.
Recall that in the `Point` and `Vector` classes you left an unimplemented constructor from `Float4` as well. These constructors have to divide by w when this makes sense or assert that it is 0 for vectors and 1 for points.

4.6 Matrix (10 Points)

We are implementing a 4x4 matrix to represent transformations in the scene. Your task is to implement the `Matrix` class.
A `Matrix` class can be constructed by:

- the default constructor, at which point the content of the matrix is undefined.
- providing its 4 rows as `Float4` .
- through `Matrix ::zero()` which should create a matrix filled with zeroes.
- through `Matrix ::identity()`, creating the identity matrix (1 on the diagonal, 0 elsewhere)
- through `Matrix ::system(e1, e2, e3)` forming a transformation from a new coordinate system defined by the basis vectors `e1 , e2, e3`.

The bracket operator `Matrix ::operator[]` should allow you to access a particlar row of a matrix represented by a `Float4` . Note that `Float4` has its own bracket operator, so a particular matrix element can be conveniently accessed through double brackets, e.g. `M [0][3]`.
You should also implement the typical mathematical operations which are done on a matrix:

- Multiplication with a four-component vector (`Float4`), as well as with existing 3D `Point` and `Vector` and a scalar.
- `Matrix ::transpose()` should transpose the matrix.
- `Matrix ::det()` should compute the determinant of the matrix.
- `product (a, b)` should compute the product of matrices `a` and `b` .

The `Matrix` inversion function is already provided.

4.7 Instancing (40 Points)

Instancing is a method of showing the same object multiple times in the scene, without copying the object in the computer memory. Instead, the same object is referenced multiple times and a different geometric transformation is applied to each instance. Your task is to implement an `Instance` class.
The `Instance` is constructed with a pointer to another primitive — the archetype, that we are instancing. The archetype can be later reached through `Instance ::content()`.
The `Instance` conceptually transforms the archetype primitive (translation, rotation, scaling), but in reality it is not allowed to modify the archetype as it may be referenced multiple times. Instead, the total transformation T should be stored within the instance object and applied when a ray intersects it. More precisely, when a ray intersects the `Instance` object the following should happen:

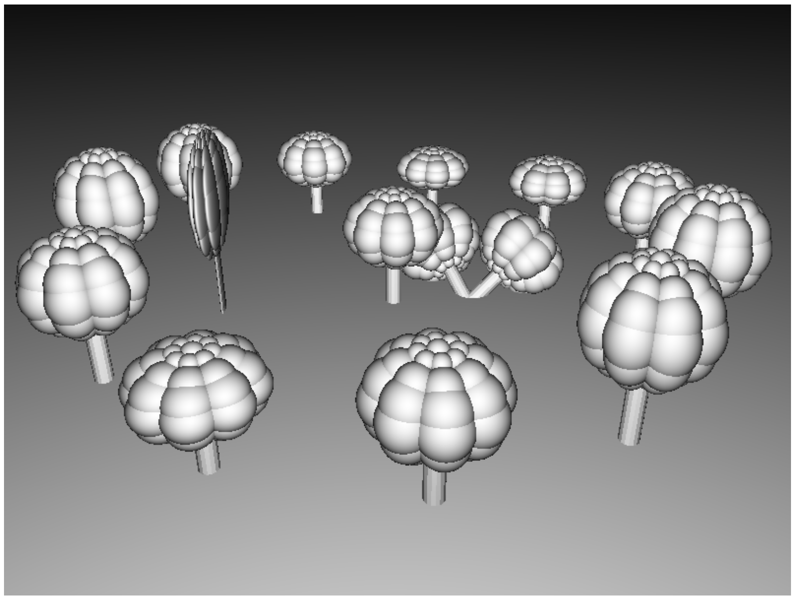

Figure 1: A simple tree model instanced multiple times with different transformations. The center tree has no transformation applied. The fallen trees to the right should origin at the same point but be differently rotated. The tree on the left is severely flattened. The 11 trees which are around are in perfect circle, but are of different height.

- The inverse tranformation T^{-1} is applied to the ray (`previousBestDistance` may need to be adjusted as well).
- The new, transformed ray is used to intersect the archetype
- The resulting `Intersection` object needs to be transformed back accordingly (distance, ray and normal).

The transformation information T is also needed when computing the bounding box of the transformed object. The computed box does not have to be tight, it is sufficient to compute the bounds of a transformed archetype's bounding box.
Note: The complete scene as provided in the `a_instancing()` function may take some time to render in debug mode. However, `scene` is just a `SimpleGroup`, making the raytracer intersect every instance and its contents. If your acceleration structure from the previous assignment is working, you may define the `scene` to use it to greatly speed up the rendering process. However, the `tree` (also `SimpleGroup`) will not benefit much from the structure as solids are heavily overlapping themselves.