Function 1 – Light Scattering

If you have questions: Martin Matuszek

Subject: Under water, the light scatters with an effect similar to fog. Your renderings should simulate this.

It's like underwater fog. Its behaviour is changing pixel colour into desired fog-colour depending on distance from camera

In your scene file, add this directly into scene:

```
1.  <Fog type="exp2" val1="0.04">
2.        <Color red="0.3" green="0.8" blue="0.98"/>
3.  </Fog>
```

```
1.  <Fog type="linear" val1="10.0" val2="20.0">
2.        <Color red="0.3" green="0.8" blue="0.98"/>
3.  </Fog>
```

- type
  - 3 possibilities "linear" "exp" "exp2"
  - it is changing how your scattering/fog will behave (you can experiment)
- val1
  - 1. for "exp" and "exp2" function, density parameter, which is changing shape of function
  - 2. for "linear", it is bottom threshold -> from which distance is fog starting to appear
- val2
  - only for "linear", it sets upper distance from camera, when everything is completely foggy
- Colour
  - colour of your fog
  - it also changes colour of background of your window

Function 2 – Textures

If you have questions: Raquel Oliveira

Subject: The seabed below the diver is made of sand, and possibly rocks, seaweed, corals... The diver's suit or equipment could also be of non-uniform appearance. Use at least one occurrence of texture mapping to render some surface details of your choice.

I created a cube with 5 faces: back, top, bottom, left and right, and put the camera in the front of the cube. This cube will be a box where the deap-sea diver should be inside (with the other objects of the animation).

I draw the cube specifying each face.

I draw a face specifying each vertex of the face (4 vertices).

To put a texture in one face of the cube, for each vertex of the face I specify the coordinate in the texture for that vertex. In this way, I map the texture for that face.

The texture is a 2D image, and it's coordinates are (x,y):

(0,1) (1,1)

(0,0) (1,0)

That's it.

Function 3 – Air bubbles (not finished…)

If you have questions: Cyril Fait or Laurent Gräbner

Subject: Propose solutions to represent and animate the air bubbles the player is generating in the water. (note that bubbles may merge to form larger ones before reaching the surface of the sea).

They are working on it.

Function 4 – Flexible tube or rope

If you have questions: Jander Nascimento or Oleg Iegorov

Subject: the diver is connected to the surface by a tube or rope which flexes as he moves or with currents. Propose a representation and physical model to animate the tube or rope.

Didn't receive anything…

Function 6 – Fish flock

If you have questions: Tigran Tchougourian

Subject: Propose a layered model to represent and animate the fishes. For each layer, define its goal and explain your choice of model among those studied in class. The last layer should be a geometric model of your choice (a mesh, a spline surface or an implicit surface), animated over time. Extend the model to get a flock of fishes flying away from the player.

First of all, the fishes will not fly away from the player.

So I'll explain the idea of the fish flock. The teacher asks to do it as a Layered model, but I only succeed for the behavior model. But there are a lot of them (Animating the skeleton, Skin deformation, clothes, hair and so on…).

XML code

First let's have a look of the xml code.

```
1.  <FishFlock population="50" xmin="-100" xmax="100" ymin="-100" ymax="100"
2.  zmin="-100" zmax="100">
3.        <Fish mass="20">
4.              <Material>
5.                    <DiffuseColor red="0.1" green="0.1" blue="0.8" />
6.                    <SpecularColor red="0.2" green="0.3" blue="0.2"
7.  exponent="2" />
8.                    <ReflectiveColor red="0.4" green="0.2" blue="0.2" />
9.              </Material>
10.       </Fish>
```

```
11.  </FishFlock>
```

So basically you can create as many fishflock as you want, but they will not communicate together (that mean they will collide).

The <FishFlock/> tag has 2 attributes:

- population : how many fishes do you want to appear on the scene
- Bounding box : a close space were the fishflock will live.
  - [xmin; xmax] : represent the limit on x coordinates.
  - [ymin; ymax] : represent the limit on y coordinates.
  - [zmin; zmax] : represent the limit on z coordinates.

For example if you have some water until z = 0 then you have to put a bounding limit to zmax = 0. By default the values are set to [-32; 32].

Then you notice that inside the <FishFlock/> tag there is a <Fish/> tag. This unique tag represents how your fish will look like. They use the same principle as for sphere or cylinder. So you just need to put a material and they will take this colour. Of course you are free (and I encourage you) to change the model in the fish.cpp (method draw()) to have a real fish instead of an "air plane".

How the code works

The code is quite simple, because it follows a simple idea. The fishflock manage a list of fishes. Each fish has a position, a velocity (bounded to a speed_max), a mass (and should have a direction for animating but not implemented). Each time animation is called, the fishflock browse all it fishes and apply a set of rules.

Pseudo-code:

```
1.  Animate() {
2.        Vector V1, V2, V3
3.        For each Fish f
4.              V1 = rule1(f)
5.              V2 = rule2(f)
6.              V3 = rule3(f)
7.
8.              f.velocity = f.velocity + V1 + V2 + V3
9.              f.position = f.position + f.velocity
10.       end
11. }
```

These rules are very important because they determine the behaviour of the fish, and consequently the fishflock behaviour. There are 3 main rules (lecture) to apply to have a basic behaviour.

- Attraction towards an object: in the code I call it cohesion, this rule calculate the center of the fishflock and attract the current fish to this position
- Avoid inter-collision : in the code I call it separation, this rule avoid collision thanks to a threshold. So if two fishes are to close, then the rule calculate an vector which will consequently be opposite to the cohesion rule.
- Alignment: this rule is specific to the fishflock, because the move together as one big fish. It calculate for each fish the average velocity of all fishes, and thanks to a formula:

$$(Current\_fish\_velocity - average\_fishes\_velocity) / 8$$

they will try to acquire the same velocity (but never succeed otherwise it not a fish flock)

There is two other rules which just let you enjoy their swim without moving the camera. Bounding box and bounding velocity. Te first one avoid the fishflock to go out of the bounding radius of opengl, and the second avoid fish to move to fast. Try to comment it and you'll see it's not as cool (fishflock.cpp inside method animate()).

I also implement other rules just to try thing, like chaos(). It basically, at a given time returns a huge vector (opposite to the cohesion) and all the fishes are spreading around the scene. It can be used if you want to add a predator in the scene, so the fishflock will spread if it comes to close.

So hope you'll enjoy it. I which I could spend more time, and add some obstacles (very easy just takes time).

BTW: this is a known algorithm, it calls Boids algorithm so quote it during your defence!