Function Representation of the 3D objects

Aim: The study of the process of the geometric modeling with using the Function Representation of 3D models.

Task: Create 3D models using the geometric modeling language and system HyperFun.

Result: The source code. The report.

Theoretical part:

HyperFun is a simple geometric modeling language for F-rep objects. F-rep stands for function representation. In F-rep, objects are described using a single real continuous function F(x1, x2, x3, ..., xn) >= 0. (Website of project: http://hyperfun.org)

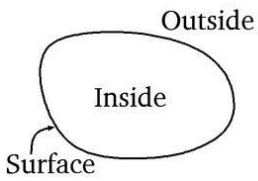
Since F-rep models are more general than traditional skeletal implicit surfaces, convolution surfaces, distance-based models, CSG (Constructive Solid Geometry), sweeps, and voxel models, HyperFun can deal with all these geometric models.

In HyperFun, F-rep objects are described using assignments, conditional selections, and iterations as in traditional programming languages. In addition to arithmetic and relational operators, HyperFun has built-in set-theoretic operators such as union, intersection, subtraction, and so on. Some predefined library primitives and operators are available in HyperFun. Using these functionalities, quite complex objects can be modeled.

F-rep

In F-rep (Function Representation), a geometric object is defined by a single real continuous function F(x1, x2, x3, ..., xn) >= 0. Let's think why F(x1, x2, x3, ..., xn) >= 0 can represent an object.

A geometric object in 3D space is described by: the inside, the surface, and the outside.



We assume that F(x1, x2, x3, ..., xn) >= 0 is a rule which determines where a given point belongs, that is:

F(x1, x2, x3, ..., xn) > 0 : the inside of an object

F(x1, x2, x3, ..., xn) = 0 : the surface of an object

F(x1, x2, x3, ..., xn) < 0 : the outside an object

To make a mesh of triangles (polygonize) on the surface of an F-rep object, the function has to be sampled in space. The bounding box is used to limit a range of sampling in space, and it is divided into fixed-size grids. The function is sampled in each node of the grid; then polygons are generated based on the sampled values of the function. The high grid density makes a good approximation of an object, but the number of polygons increases.

Geometric modeling using HyperFun

We begin with a simple sphere. The program example1.hf describes a solid sphere:

$$5^2 - \left(x^2 + y^2 + z^2\right) \geqslant 0$$

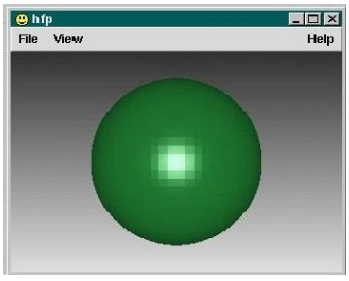
```
-- example1.hf


my_model(x[3], a[1])
{
  my_model = 5^2 - (x[1]^2 + x[2]^2 + x[3]^2);
}
```

To see an object, the program has to be interpreted by visualization tools. We use the **HyperFun Polygonizer** that visualizes F-rep objects described in the HyperFun program with polygons.

```
 hfp sphere.hf
```

The following window will be shown, and you can see the resulting polygonized object. You can rotate an object with the left mouse button and scale with the right mouse button. Select Quit in the File menu to quit the program.



Let’s see the program step by step.

```
line 1: -- sphere.hf

line 2:

line 3: my_model(x[3], a[1])

line 4: {

line 5:   my_model = 5^2 - (x[1]^2 + x[2]^2 + x[3]^2);

line 6: }
```

Line 1: The first line is a comment. The section from – to the newline is assumed to be a comment. You can insert comments anywhere in a program.

Line 3: The HyperFun program begins with the object name my_model(x[3], a[1]), x and a are arrays. Note that x and a are reserved key words in HyperFun, so they can be used only in a special way: x is a point coordinates array and a is used to pass external values. In this example, a is not used. In HyperFun, an array index begins with 1. As defaults, x[1], x[2] and x[3] are mapped to point coordinates as:

```
x[1] -> x

x[2] -> y

x[3] -> z
```

Line 4, 6: The body of my_model is described in the block enclosed between { and }. Each statement must be terminated by a semicolon.

Line5: 5^2 - (x[1]^2 + x[2]^2 + x[3]^2) represents $5^2 - (x^2 + y^2 + z^2)$. The following table shows arithmetic operators available in HyperFun.

Arithmetic operators in HyperFun	
+	addition
-	subtraction
*	multiplication
/	division
^	power

The object my_model has to return a function value; to do so my_model = is used. You can see later that local variables can be used in HyperFun; a variable which has the same name as an object is a special local variable used to return a value. This is like return used in C. The Polygonizer uses such values to visualize the objects.

Use of library functions

In the next example, we use a predefined library function available in HyperFun to model a sphere.

```
-- example2.hf


my_model(x[3], a[1])
{
  array center[3];
  center = [0.0, 0.0, 0.0];
  sphere = hfSphere(x, center, 5.0);
  my_model = sphere;
}
```

Polygonize the above model with the command:

```
hfp example2.hf
```

The resulting object is the same as the first example.

Let’s examine the program. The structure is almost the same as the first example, so only new things are explained here.

```
line  1: -- example2.hf
line  2:
line  3: my_model(x[3], a[1])
line  4: {
line  5:   array center[3];
line  6:
line  7:   center = [0.0, 0.0, 0.0];
line  8:   sphere = hfSphere(x, center, 5.0);
line  9:   my_model = sphere;
```

```
line 10: }
```

Line 5: array center[3] is a declaration of an array. An array must be declared at the beginning of an object's block before being used. The keyword array is used to declare arrays. In this case, center is declared as an array with size 3.

Line 7: The array center is initialized. There are two ways to initialize arrays: simultaneously and individually.

```
-- Initialize an array simultaneously

center = [0.0, 0.0, 0.0];

-- Initialize an array individually

center[1] = 0.0;

center[2] = 0.0;

center[3] = 0.0;
```

Line 8: hfSphere is a library function for a solid sphere. You can think of hfSphere as a black box that can represent a solid sphere, so you only specify its position and radius. HyperFun has such library functions and operations, see F-rep library. Here, sphere is a local variable to save the value which hfSphere returns. The only type of variables used in HyperFun is real, so you can use variables without declarations.

Line 9: Finally, to return a value, my_model = sphere is used.

Tasks:

Modelling Implicit Surfaces Using Equations

1. In the text editor type the following model of an ellipsoid:

```
test(x[3],a[1]){

test = 1-(x[1]/3)^2 -(x[2]/5)^2 -(x[3]/7)^2;

}
```

2. Save the model to the tst.hf text file in the HyperFunPolygonizer (HPF) folder. Use the HyperFunPolygonizer to render the model with the command:

```
hfp tst.hf
```

3. Change the half-axes of the ellipsoid in the equation and get new surface. If you see trimmed ellipsoid you need to increase the bounding box size for the entire scene, for example:

```
hfp tst.hf –b 10,12,15
```

4. Type the model of an ellipsoid with parameters:

```
test(x[3],a[3]) {
test = 1-(x[1]/a[1])^2 -(x[2]/a[2])^2 -(x[3]/a[3])^2;
}
```

You can change parameter values with the command:

```
hfp tst.hf –a 5,7,9
```

5. Use the lecture notes and model a torus with the equation and with parameters. Render the torus with HFP.

FRep Library Primitives

1. Type the following model of an ellipsoid primitive:

```
test(x[3],a[1]){

array center [3];

center=[0,0,0];

test = hfEllipsoid(x,center,6,2,2);

}
```

2. Save the model to the tst.hf text file in the HFP folder. Use the HyperFun Polygonizer to render the model with the command:

```
hfp tst.hf
```

3. Type the following model of an ellipsoid primitive with parameters:

```
test(x[3],a[3]){

array center [3];

center=[0,0,0];

test = hfEllipsoid(x,center,a[1],a[2],a[3]);

}
```

You can change parameter values with the command:

```
hfp tst.hf –a 5,7,9
```

Try different half-axes by changing parameters.

4. Make a model of a superellipsoid with two parameters in the array a[2] using the F-rep library primitive. Input different parameter values and render different superellipsoids

Deformations

1. In the text editor type the following model of a superellipsoid primitive:

```
test(x[3],a[1]){

array center [3];

center=[0,0,0];

test = hfSuperell(x, center, 6, 2, 2, 0.2, 0.2);

}
```

2. Save the model to the tst.hf text file in the HFP folder. Use the HyperFun Polygonizer to render the model with the command:

```
hfp tst.hf
```

3. Change the model by including a twisting operation:

```
test(x[3],a[1]){

array center [3];

array xt[3];

xt[1]=x[1];

xt[2]=x[2];

xt[3]=x[3];

tmp = hfTwistX(xt,-6,6,0,-3.14);

center=[0,0,0];

test = hfSuperell(xt,center,6,2,2,0.2,0.2);

}
```

Pay attention to the new xt argument of the hfSuperEll primitive!

4. - Render the model with HFP.

- Try different parameters of the twisting operation

Example of loop operation:

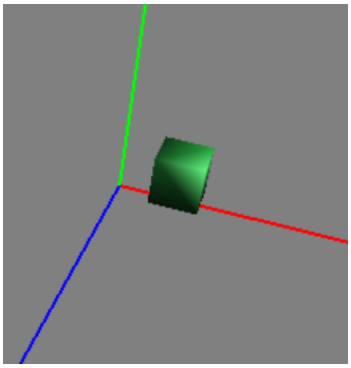
```
j=1;
while (j <= 3) loop
j = j + 1;
endloop;
```

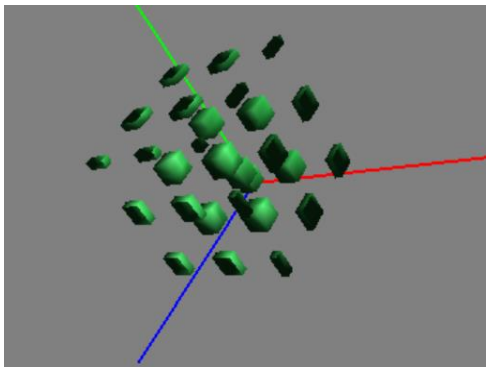
Individual tasks:

Create unit-cell and construct array 3x3x3 of this elements by repetition

Example:

Unit-cell: Array:





Variant 1:

Unit-cell: block (rectangle)

Variant 2:

Unit-cell: prizm (triangle)

Variant 3:

Unit-cell: Sphere

Variant 4:

Unit-cell: Ellipsoid

Variant 5:

Unit-cell: Torus

Variant 6:

Unit-cell: Cone