Listing 1. Sample of main CSG function with virtual methods

```
  TFunctionCSG = class                                    // main class of the algorithm
protected
    FMode: TFunctionMode;                                 // sets out the basic shapes
and                                                       operations on shapes          public
    function CopyObj:TFunctionCSG; virtual;               //  allow to make a copy of the
object                                                    and his subobjects

    function GetModeName: AnsiString; virtual;            // can download the object name
                                                          of the class type. For example
                                                          the shape of a sphere in a class
                                                          will return the value of editor
                                                          the 'sphere'. Information is
                                                          presented in the editor in a
                                                          given tree-building production
                                                          (Fig. 4 The scene tree)

    function Distance(const aPosition:Tv3D): Single;      // function of the distance which
                                                          determine the shapes (the merits
                                                          of the algorithm)
  virtual;
  procedure SetConnectors();  virtual;                    // override this feature allow
                                                          to create own connection points
                                                          for the shape.  Each shape has
                                                          declared another type of points.

  procedure AddConnector(aNormal,aPosition:Tv3D);         //  creating a point of combining
                                                          objects in space by specifying
                                                          parameters of the position and
                                                          direction of the merger (X, Y, Z)

  procedure LoadObj(var aFile: file);                     //read information about the shape

  procedure SaveObj(var aFile: file);                     //record information about the
                                                          shape
  … /*rest of the code (if needed) */
  end;

// declaration of basic shapes and operations using Fmode parameter

    fmEmpty = 0;                                          // empty shape
    fmOperation = 1;                                      // sum
    fmMorfing = 2;                                        // morphing
    fmSphere = 3;                                         // sphere
    fmBox = 4;                                            // box
    fmTorus = 5;                                          // torus
    fmPyramid = 6;                                        // pyramid
    fmCylinder = 7;                                       // cylinder
```

Listing 2. Morphing and Boolean operations

```
// morphing operation
// FFactor – morphing parameter
// result – designate morphing of two functions FFunctionA and FFunctionB
// Distance – distance vector
// aPosition – point in 3D space, which is sampled
// v3d_sub – subtraction vector
// FPostion – point in the middle of a shape described by function

function TMorfingCSG.Distance(const aPosition:Tv3D): Single;
var
  pos : Tv3D;
begin
  pos := v3d_sub(aPosition, FPosition);
```

```
  result:= FFunctionA.Distance(pos)*(1-FFactor)+FFunctionB.Distance(pos)* FFactor;
end;
-----------------------------------------------------------------------------------------

// union operation
// result – designate the sum of two functions FFunctionA and FFunctionB
// Distance – distance vector
// aPosition – point in 3D space, which is sampled

function TOperationCSG.Union(const aPosition: Tv3D): Single;
begin
  result:=get_max(FFunctionA.Distance(aPosition),FFunctionB.Distance(aPosition));
end;
-----------------------------------------------------------------------------------------

// difference operation
// result – designate the difference of two functions FFunctionA and FFunctionB
// Distance – distance vector
// aPosition – point in 3D space, which is sampled

function TOperationCSG.Difference(const aPosition: Tv3D): Single;
begin
  result:=get_min(FFunctionA.Distance(aPosition),-FFunctionB.Distance(aPosition));
end;
-----------------------------------------------------------------------------------------

// intersection operation
// result – designate the intersection of two functions FFunctionA and FFunctionB
// Distance – distance vector
// aPosition – point in 3D space, which is sampled

function TOperationCSG.Intersection(const aPosition: Tv3D): Single;
begin
  result :=get_min(FFunctionA.Distance(aPosition),FFunctionB.Distance(aPosition));
end;
```

Listing 3. Definition basic shapes using function description

```
// sphere
// FRadius – sphere radius

function TSphereCSG.Distance(const aPosition:Tv3D): Single;
begin
    result := -(v3d_Distance(aPosition,FPosition)-FRadius);
end
-----------------------------------------------------------------------------------------

// box
// Tv3D – vector in 3D space with starting point in the beginning of coordinate system

function TBoxCSG.Distance(const aPosition:Tv3D): Single;
var
  sabs:Tv3D;
begin
   sabs := v3d_Sub(v3d_abs(v3d_sub(aPosition, FPosition)), v3d_Scale(FSize,0.5));
   result := -get_max(sabs.y, get_max(sabs.x, sabs.z));
end;
-----------------------------------------------------------------------------------------

// cylinder
// FRadius – cylinder radius

function TCylinderCSG.Distance(const aPosition:Tv3D): Single;
var
  dir : tv3D;
  sabs:single;
begin
```

```
  dir := v3d_Sub(aPosition,FPosition);
  sabs := abs(dir.y)-FHeight*0.5-0.01;
  dir.y := 0;
  result := get_min(-sabs,-(v3d_Length(dir)-FRadius));
end;
```

Listing 4. Creating a sphere shape

```
// sphere shape
// FRadius – sphere radius
// FBBox.vMin – minimum cuboid (sampling area)
// FBBox.vMax – maximum cuboid (sampling area)

constructor TSphereCSG.Create;
begin
  FMode := fmSphere;
  FRadius := 1;
  FBBox.vMin := v3d(-FRadius,-FRadius,-FRadius);
  FBBox.vMax := v3d(FRadius, FRadius, FRadius);
end;
```

Listing 5. Example of assembling two functions

```
Distance = Operation.Distance (TSphereCSG.Distance(), TBoxCSG.Distance())
```

Listing 6. Switching points for the mesh

```
// combining shapes
// FNormal – determines the direction of connection
// FPositionLocal – specifies the position vector in local coordinates
// FPositionGlobal – specifies the position vector in global coordinates
// Fused – determines whether the connection point has been already used and whether you can
join it

TConnectorCSG = record
   FNormal: Tv3D;
   FPositionLocal : Tv3D;
   FPositionGlobal : Tv 3D;
   FUsed : boolean;
   end;
…
procedure TFunctionCSG.AddConnector(aNormal,aPosition:Tv3D);
begin
  Inc(FConnectorCount);
  SetLength(FConnectorList,FConnectorCount);

  with FConnectorList[FConnectorCount-1] do
  begin
    FNormal := aNormal;
    FPositionLocal := aPosition;
    FPositionGlobal := aPosition;
    FUsed := false;
  end;
end;
```

Listing 7. The cubes located on the surface of the shape described by a function

```
// mesh model generation
FModel.BeginCreate;
```

```
  if FFunctionObject <> nil then
  begin

// drawing mesh triangles
    glBegin(GL_TRIANGLES);
    minX := Round(FFunctionObject.BBox.vMin.x * FGridSize) - 1;
    minY := Round(FFunctionObject.BBox.vMin.y * FGridSize) - 1;
    minZ := Round(FFunctionObject.BBox.vMin.z * FGridSize) - 1;
    maxX := Round(FFunctionObject.BBox.vMax.x * FGridSize);
    maxY := Round(FFunctionObject.BBox.vMax.y * FGridSize);
    maxZ := Round(FFunctionObject.BBox.vMax.z * FGridSize);

    for px := -2 to 1 do
    for px := minX to maxX do
    begin
      if ProgressBar1 <> nil then
        ProgressBar1.Position := Round(100 * (px - minX) / (maxX - minX));
      for py := minY to maxY do
        for pz := minZ to maxZ do

// drawing boxes (sampling cubes) on the solid surface
          DrawSimpleBox(px, py, -pz - 1);
    end;

// exiting and mesh compilation
    FModel.EndCreate;
    FGeneratingTime := GetTickCount()- fnow;
end;
```

Listing 8. Declaration of parameters for the samples (cubes)

```
Type

// point on the grid in 3D space
PGridPoint = ^TGridPoint;

// declaration pointer to point
TGridPoint = record

// point position
    Pos: Tv3D;

// normal vector at the point
    Normal: Tv3D;

// distance from the point to the plane
    Value: Single;
  end;

// grid on which the point is located – the cube have 8 vertices
TGridCube = record

// array of points of the cube
   GridPoint: Array [0 .. 7] of PGridPoint;
  end;
```

Listing 9. Checking cubes and points

```
/*rest of the code (if needed) */
// checking whether the cube is outside the object
  if edgeTable[CubeIndex] = 0 then
    exit;
 …
// testing whether all points are inside or outside the object
  if (inside = 0) or (inside = 8) then
    exit;
/*rest of the code (if needed) */
```

Listing 10. Edge interpolation

```
with GridCube do

    if (edgeTable[CubeIndex] and 001) <> 0 then
      Interpolate(GridPoint[0]^, GridPoint[1]^, VertList[0], Norm[0]);
    if (edgeTable[CubeIndex] and 002) <> 0 then
      Interpolate(GridPoint[1]^, GridPoint[2]^, VertList[1], Norm[1]);
    if (edgeTable[CubeIndex] and 004) <> 0 then
      Interpolate(GridPoint[2]^, GridPoint[3]^, VertList[2], Norm[2]);
    …
    /*rest of the code (if needed) */
    …
    if (edgeTable[CubeIndex] and 2048) <> 0 then
      Interpolate(GridPoint[3]^, GridPoint[7]^, VertList[11], Norm[11]);
end
```

FIGURE 3

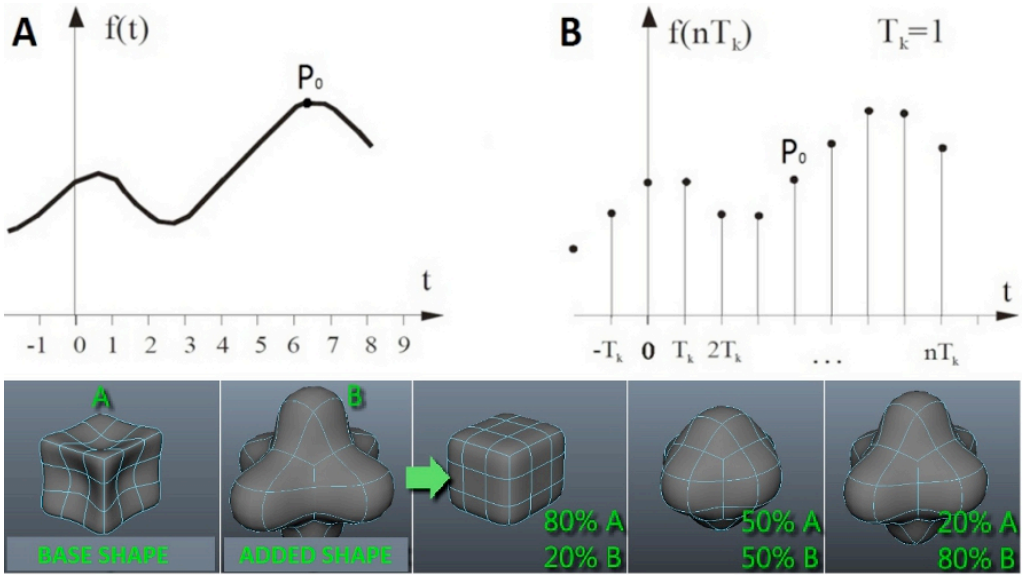


FIGURE 4

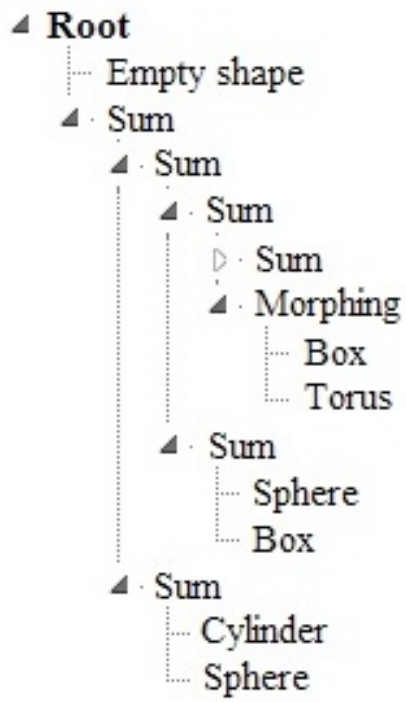
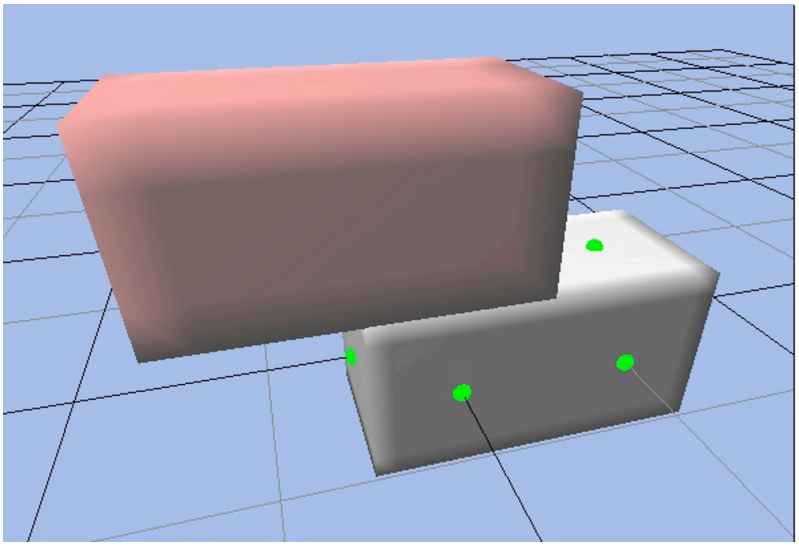


FIGURE 5



N = (0, 1, 0) , P = (0, sy, 0)
N = (0, -1, 0) , P = (0, -sy, 0)
N = (-1, 0, 0) , P = (-sx, 0, 0)
N = (1, 0, 0) , P = (sx, 0, 0)
N = (0, 0, 1) , P = (0, 0, sz)
N = (0, 0, -1) , P = (0, 0, -sz)
N = (0, 0, 0) , P = (0, 0, 0)

FIGURE 7

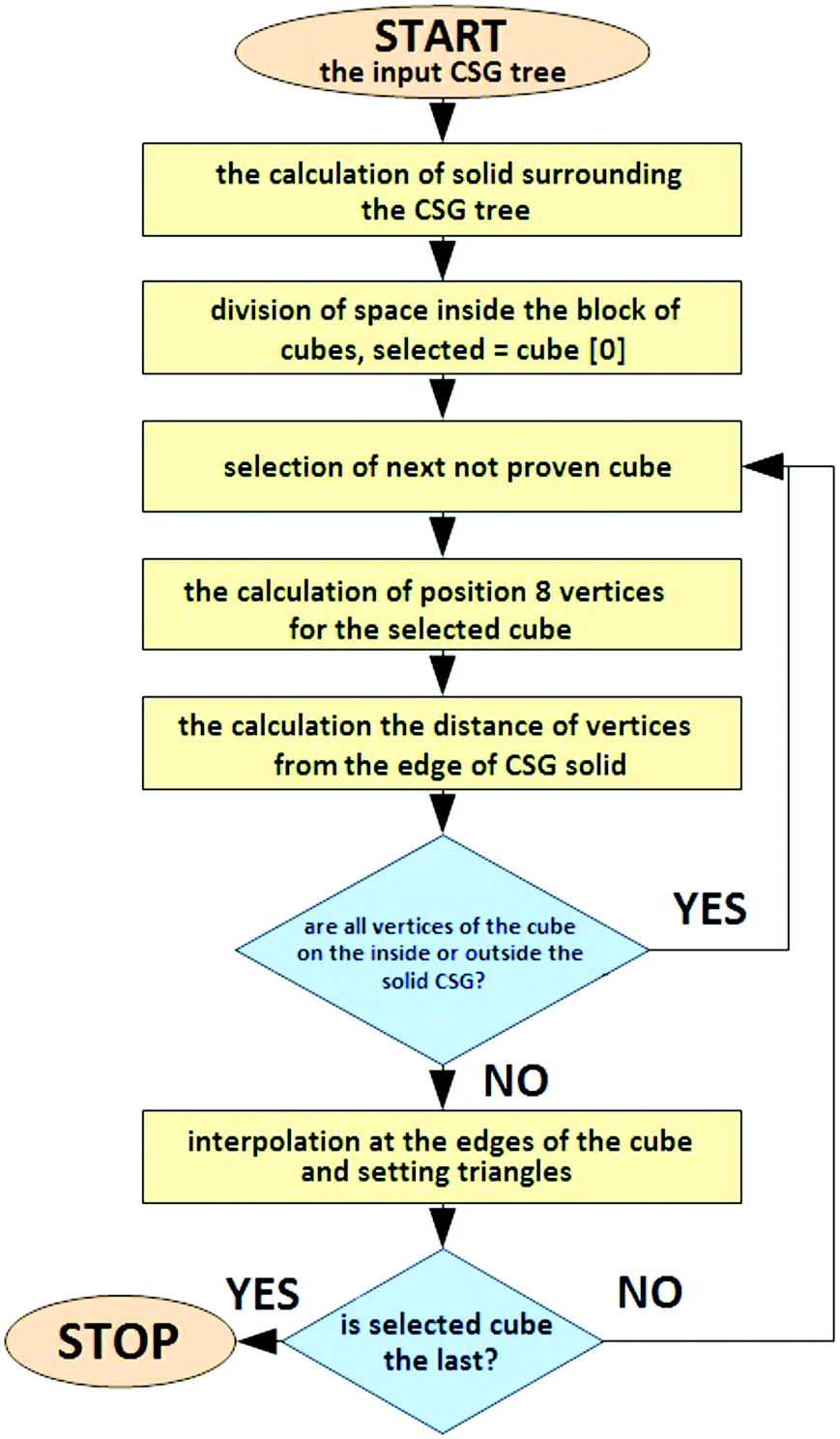
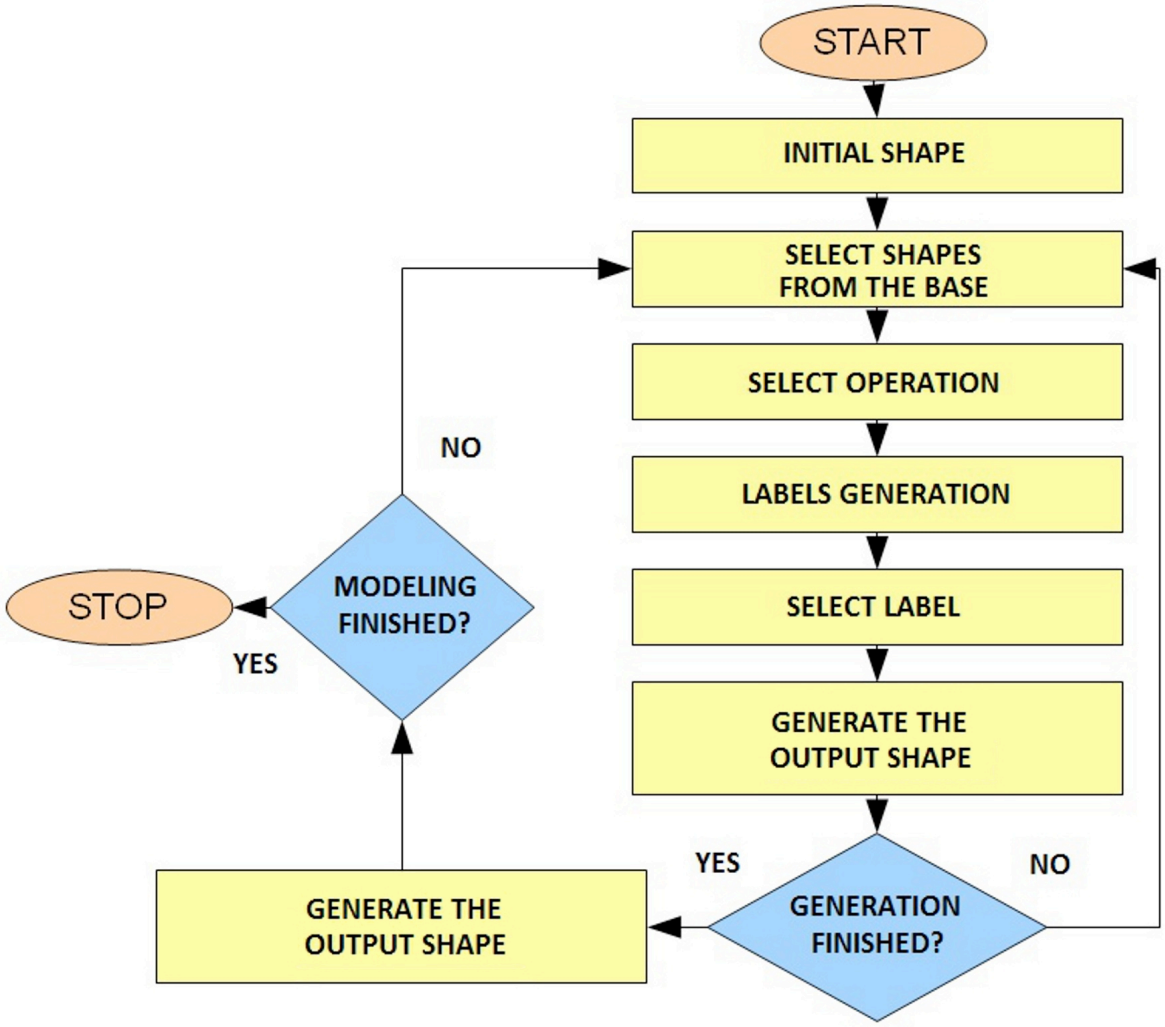

START
the input CSG tree
the calculation of solid surrounding the CSG tree
division of space inside the block of cubes, selected = cube [0]
selection of next not proven cube
the calculation of position 8 vertices for the selected cube
the calculation the distance of vertices from the edge of CSG solid
are all vertices of the cube on the inside or outside the solid CSG?
YES
NO
interpolation at the edges of the cube and setting triangles
is selected cube the last?
YES
NO
STOP

FIGURE 11


START
INITIAL SHAPE
SELECT SHAPES FROM THE BASE
SELECT OPERATION
LABELS GENERATION
SELECT LABEL
GENERATE THE OUTPUT SHAPE
GENERATION FINISHED?
YES
NO
GENERATE THE OUTPUT SHAPE
MODELING FINISHED?
NO
YES
STOP