Maillages.

Laurent Cancé Francis
14/09/2017

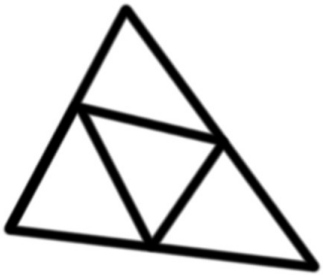
I. Procédé

Dans l'élaboration de maillages, il existe une fonction utile qui « arrondie » le volume selon la densité de maillage existant. Communément nommée « nurbs », cette méthode de génération de maillage est simplifiable et je présente ici, un algorithme évident d'ajout de triangles.

L'idée est de partitionner le maillage existant en utilisant une moyenne de positionnement:

II. Simplification A

Avec :
Vertices[n]
Faces[n] .v0 .v1 .v2 et v[i] pointeur sur les Vertices



l'algorithme générant 4 faces par population donne :

```
int v0,v1,v2;
int m0,m1,m2,m;

res=new Object3D;

EdgesInit(obj,nFaces*3);
```

```
for (n=0;n<nFaces;n++)
{
		AddEdge(Faces[n].v0,Faces[n].v1);
		AddEdge(Faces[n].v1,Faces[n].v2);
		AddEdge(Faces[n].v2,Faces[n].v0);
}
// EdgesLength = num list sans doublons

res->Init(nVertices+EdgesLength,nFaces*4);

for (n=0;n<nVertices;n++)
{
		res->Vertices[n]=Vertices[n];
}

for (n=0;n<EdgesLength;n++)
{
		res->Vertices[nVertices+n]=(Vertices[Edges[n].b]+Vertices[Edges[n].a])*0.5f;
		res->VerticesMap[nVertices+n]=(VerticesMap[Edges[n].b]+VerticesMap[Edges[n].a])*0.5f;
}

nn=0;
for (n=0;n<nFaces;n++)
{
		v0=Faces[n].v0;
		v1=Faces[n].v1;
		v2=Faces[n].v2;

		m0=nVertices+WitchEdge(v0,v1);
		m1=nVertices+WitchEdge(v1,v2);
		m2=nVertices+WitchEdge(v2,v0);

		m=nVertices+EdgesLength+n;

		res->Faces[nn].v0=v0;
		res->Faces[nn].v1=m0;
		res->Faces[nn].v2=m2;
		nn++;

		res->Faces[nn].v0=m0;
		res->Faces[nn].v1=v1;
		res->Faces[nn].v2=m1;
		nn++;

		res->Faces[nn].v0=m1;
		res->Faces[nn].v1=v2;
		res->Faces[nn].v2=m2;
		nn++;

		res->Faces[nn].v0=m0;
		res->Faces[nn].v1=m1;
		res->Faces[nn].v2=m2;
		nn++;
}

for (n=0;n<res->nVertices;n++) VerticesTmp[n]=res->Vertices[n];

for (k=0;k<2;k++)
{
		for (n=0;n<EdgesLength;n++)
		{
			res->Vertices[nVertices + n]=medium(res,nVertices + n);
		}

		for (n=0;nVertices;n++)
		{
			res->Vertices[n]=medium(res,n);
```

```
                }
        }

        for (n=0;n<res->nVertices;n++) res->Vertices[n].tag=0;

        for (n=0;n<res->nVertices;n++)
         if (res->Vertices[n].tag==0)
         {
                int nb=0;
                CVector G;
                G.Init(0,0,0);

                for (k=0;k<res->nVertices;k++)
                        if (res->Vertices[k].tag==0)
                        {
                                if (Close(res,n,k))
                                {
                                        res->Vertices[k].tag=-1;
                                        G+=res->Vertices[k];
                                        nb++;
                                }
                        }

                G=G/nb;

                for (k=0;k<res->nVertices;k++)
                        if (res->Vertices[k].tag<0)
                        {
                                res->Vertices[k].tag=1;
                                res->Vertices[k]=G;
                        }
         }
```

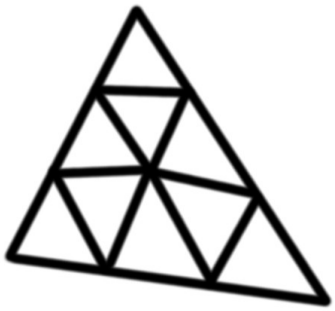
avec :

```
CVector medium(*obj*,int nv)
{
        int n;
        int nb=0;
        CVector P;

        for (n=0;n<nVertices;n++) Vertices[n].tag=0;

        for (n=0;n<nFaces;n++)
        {
                if (Close(Faces[n].v0,nv)) { Faces[n].v[1]->tag=1; Faces[n].v[2]->tag=1; }
                if (Close(Faces[n].v1,nv)) { Faces[n].v[0]->tag=1; Faces[n].v[2]->tag=1; }
                if (Close(Faces[n].v2,nv)) { Faces[n].v[0]->tag=1; Faces[n].v[1]->tag=1; }
        }

        nb=0;
        P.Init(0,0,0);

        for (n=0;n<nVertices;n++)
         if (Vertices[n].tag)
         {
                P+=Vertices[n];
                nb++;
         }

        P=P/nb;

        return P;
}
```

et :

```
bool Close(*obj*,int a,int b)
{
	CVector u;
	u=Vertices[a]-Vertices[b];
	if (u.Norm()<SMALLF) return true;
	else return false;
}
```

III. Génération B : 9 triangles par population.



```
	EdgesInit(obj,nFaces*3);

	for (n=0;n<nFaces;n++)
	{
		AddEdge(Faces[n].v0,Faces[n].v1);
		AddEdge(Faces[n].v1,Faces[n].v2);
		AddEdge(Faces[n].v2,Faces[n].v0);
	}
	// EdgesLength = num list sans doublons

	res->Init(nVertices+EdgesLength*2+nFaces,nFaces*9);

	for (n=0;n<nVertices;n++)
	{
		res->Vertices[n]=Vertices[n];
	}

	for (n=0;n<EdgesLength;n++)
	{
		res->Vertices[nVertices+n*2+0]=Vertices[Edges[n].a] + (Vertices[Edges[n].b]-Vertices[Edges[n].a])*1/3;
		res->Vertices[nVertices+n*2+1]=Vertices[Edges[n].a] + (Vertices[Edges[n].b]-Vertices[Edges[n].a])*2/3;
	}

	for (n=0;n<nFaces;n++)
	{
		res->Vertices[nVertices+EdgesLength*2+n]=(Faces[n].v[0] + Faces[n].v[1] + Faces[n].v[2])/3;
	}

	nn=0;
	for (n=0;n<nFaces;n++)
	{
		v0=obj->Faces[n].v0;
		v1=obj->Faces[n].v1;
		v2=obj->Faces[n].v2;

		m0=WitchEdge(v0,v1);
```

```
m1=WitchEdge(v1,v2);
m2=WitchEdge(v2,v0);

if (Close(obj,Edges[m0].a,v0))
{
	ma01=nVertices+2*m0 + 0;
	mb01=nVertices+2*m0 + 1;
}
else
{
	ma01=nVertices+2*m0 + 1;
	mb01=nVertices+2*m0 + 0;
}

if (Close(obj,Edges[m1].a,v1))
{
	ma12=nVertices+2*m1 + 0;
	mb12=nVertices+2*m1 + 1;
}
else
{
	ma12=nVertices+2*m1 + 1;
	mb12=nVertices+2*m1 + 0;
}

if (Close(obj,Edges[m2].a,v2))
{
	ma20=nVertices+2*m2 + 0;
	mb20=nVertices+2*m2 + 1;
}
else
{
	ma20=nVertices+2*m2 + 1;
	mb20=nVertices+2*m2 + 0;
}

m=nVertices+2*EdgesLength+n;

res->Faces[nn].v0=v0;
res->Faces[nn].v1=ma01;
res->Faces[nn].v2=mb20;
nn++;

res->Faces[nn].v0=mb20;
res->Faces[nn].v1=ma01;
res->Faces[nn].v2=m;
nn++;

res->Faces[nn].v0=m;
res->Faces[nn].v1=ma01;
res->Faces[nn].v2=mb01;
nn++;

res->Faces[nn].v0=m;
res->Faces[nn].v1=mb01;
res->Faces[nn].v2=ma12;
nn++;

res->Faces[nn].v0=ma12;
res->Faces[nn].v1=mb01;
res->Faces[nn].v2=v1;
nn++;

res->Faces[nn].v0=mb20;
res->Faces[nn].v1=m;
res->Faces[nn].v2=ma20;
nn++;
```

```
		res->Faces[nn].v0=ma20;
		res->Faces[nn].v1=m;
		res->Faces[nn].v2=mb12;
		nn++;

		res->Faces[nn].v0=mb12;
		res->Faces[nn].v1=m;
		res->Faces[nn].v2=ma12;
		nn++;

		res->Faces[nn].v0=mb12;
		res->Faces[nn].v1=v2;
		res->Faces[nn].v2=ma20;
		nn++;

}

for (k=0;k<2;k++)
{
		for (n=0;n<EdgesLength;n++)
		{
			res->Vertices[nVertices + n*2 +0]=medium(res,nVertices + n*2 +0);
			res->Vertices[nVertices + n*2 +1]=medium(res,nVertices + n*2 +1);
		}

		for (n=0;n<nVertices;n++)
		{
			res->Vertices[n]=medium(res,n);
		}

		for (n=0;n<obj->nFaces;n++)
		{
			res->Vertices[nVertices+EdgesLength*2+n]=medium(res,nVertices+EdgesLength*2+n);
		}
}

for (n=0;n<res->nVertices;n++) res->Vertices[n].tag=0;

for (n=0;n<res->nVertices;n++)
if (res->Vertices[n].tag==0)
{
		int nb=0;
		CVector G;
		G.Init(0,0,0);
		for (k=0;k<res->nVertices;k++)
			if (res->Vertices[k].tag==0)
			{
				if (Close(res,n,k))
				{
					res->Vertices[k].tag=-1;
					G+=res->Vertices[k];
					nb++;

				}
			}

		G=G/nb;

		for (k=0;k<res->nVertices;k++)
			if (res->Vertices[k].tag<0)
			{
				res->Vertices[k].tag=1;
				res->Vertices[k]=G;
			}
}
```