Modelli MilkShape 3D e OpenGL ES

In questo articolo spiegherò come usare i modellio di MilkShape3D nelle applicazioni per sistemi embedded che fanno uso delle OpenGL ES, tra essi ad esempio cellulari o palmari.
Questo tutorial è basato sull'ormai famoso codice reperibile a questo sito:
http://rsn.gamedev.net/tutorials/ms3danim.asp
Inizierò analizzando la semplicissima classe che conterrà poi tutti i dati della mesh stessa.

```
class Mesh : public CBase
{

public:

	//Class Constructor
	Mesh()
	{
		delete iVertex;
		delete iFace;

		delete[] iGLVerts;
		delete[] iGLNormals;
		delete[] iGLTexCoords;
		delete[] iGLTris;

		m_numJoints = 0;
		m_pJoints = NULL;

		m_Time = 0;
		m_totalTime = 0;
		m_looping = true;
		skeletoned = false;

		restart();
	}

	//Class Deconstructor
	~Mesh()
	{
		delete iVertex;
		delete iFace;

		delete[] iGLVerts;
		delete[] iGLNormals;
		delete[] iGLTexCoords;
		delete[] iGLTris;

		if( iTextureObject != 0xffffffff )
			glDeleteTextures( 1, &iTextureObject );

		TInt i=0;
		for ( i = 0; i < m_numJoints; i++ )
		{
			delete[] m_pJoints[i].m_pRotationKeyframes;
			delete[] m_pJoints[i].m_pTranslationKeyframes;
		}

		m_numJoints = 0;
```

```
            if ( m_pJoints != NULL )
            {
                  delete[] m_pJoints;
                  m_pJoints = NULL;
            }

            m_Time = 0;
            m_totalTime = 0;
            m_looping = true;
            skeletoned = false;
            //delete[] m_pTimer;

            animation = NULL;
            animActive = 0;
      }


public:

            //    Animation keyframe information
            struct Keyframe
            {
                  TInt m_jointIndex;
                  TInt m_time;      // in milliseconds
                  TReal m_parameter[3];
            };

            //    Skeleton bone joint
            struct Joint
            {
                  TReal m_localRotation[3];
                  TReal m_localTranslation[3];
                  Matrix m_absolute, m_relative;

                  TInt m_numRotationKeyframes, m_numTranslationKeyframes;
                  Keyframe *m_pTranslationKeyframes;
                  Keyframe *m_pRotationKeyframes;

                  TInt m_currentTranslationKeyframe, m_currentRotationKeyframe;
                  Matrix m_final;

                  TInt m_parent;
            };

            //    Animation Set information
            struct AnimationSet
            {
                  TInt initTime;
                  TInt endTime;
            };

public:

      // loading method
      void Load(const TReal* verts, const TInt* faces, const TReal* textures,
                        const TReal* normals, TInt nv, TInt nf, TInt textureSize
= 256 );

      // loading animation method
      void LoadAnimationData( TInt totTime, TInt nJoints, const TReal*
jointData, const TReal* keyframeData, const TInt* vertexJointData );

      // Drawing  method
```

```
void Draw();



//Some methods to position, rotate or scale the mesh in 3D space
void SetPosition( TInt x, TInt y, TInt z );
void Rotate( TInt x, TInt y, TInt z );
void Scale( TInt x, TInt y, TInt z );

// Method to set a texture to the mesh
void SetTexture( GLuint iTexture)
{
      iTextureObject = iTexture;
}

// Setting Diffuse Material
void SetDiffuseMaterial( TReal r, TReal g, TReal b, TReal a )
{
      matDiffuse[0] = (GLfloat)r;
      matDiffuse[1] = (GLfloat)g;
      matDiffuse[2] = (GLfloat)b;
      matDiffuse[3] = (GLfloat)a;
}

// Setting Ambient Material
void SetAmbientMaterial( TReal r, TReal g, TReal b, TReal a )
{
      matAmbient[0] = (GLfloat)r;
      matAmbient[1] = (GLfloat)g;
      matAmbient[2] = (GLfloat)b;
      matAmbient[3] = (GLfloat)a;
}

// Setting Specular Material
void SetSpecularMaterial( TReal r, TReal g, TReal b, TReal a )
{
      matSpecular[0] = (GLfloat)r;
      matSpecular[1] = (GLfloat)g;
      matSpecular[2] = (GLfloat)b;
      matSpecular[3] = (GLfloat)a;
}

// Setting Emission Material
void SetEmissionMaterial( TReal r, TReal g, TReal b, TReal a )
{
      matEmission[0] = (GLfloat)r;
      matEmission[1] = (GLfloat)g;
      matEmission[2] = (GLfloat)b;
      matEmission[3] = (GLfloat)a;
}

// Setting Material Shininess
void SetShininess( TInt v )
{
      shininess = v;
}
```

```
	/*
		Set the values of a particular keyframe for a particular joint.
			jointIndex		The joint to setup the keyframe for
			keyframeIndex	The maximum number of keyframes
			time			The time in milliseconds of the keyframe
			parameter		The rotation/translation values for the
keyframe
			isRotation		Whether it is a rotation or a translation
keyframe
	*/
	void setJointKeyframe( TInt jointIndex, TInt keyframeIndex, TInt time,
TReal *parameter, bool isRotation );

	/*	Setup joint matrices. */
	void setupJoints();

	/*	Set looping factor for animation. */
	void setLooping( bool looping ) { m_looping = looping; }

	/*	Advance animation by a frame. */
	void advanceAnimation();

	/*	Restart animation. */
	void restart(int n = 0 );
	void setAnimationSet(int n = 0 );

	// Mesh 3D space information
	TInt position[3];
	TInt scale[3];
	TInt rotation[3];

	// Mesh loading data
	TVertex* iVertex;
	TFace*		iFace;
	TInt		num_vertices;
	TInt		num_faces;

	//	Joint information
	TInt m_numJoints;
	Joint *m_pJoints;
	//	Total animation time
	TInt m_totalTime;
	//	Is the animation looping?
	bool m_looping;

	//	Is the mesh skeletoned
	bool skeletoned;

	// an array of different animation to initialize
	AnimationSet * animation;

	// animation information
	int animActive;
	bool nothingToDo;

	// animation time
	TInt m_Time;

	// OpenGL ES loading data
```

```
	GLuint			iTextureObject;
	GLfixed*	iGLVerts;
	GLfixed*	iGLNormals;
	GLfixed*	iGLTexCoords;
	TInt			iGLTriCount;
	TInt			iGLVertCount;
	GLushort*	iGLTris;

	// OpenGL ES Material data
	GLfloat matDiffuse[4];
	GLfloat matAmbient[4];
	GLfloat matSpecular[4];
	GLfloat matEmission[4];
	TInt shininess;

};
```

Per prima cosa analizziamo il metodo Load(). Questo prende come parametri di input alcuni array di differenti tipi salvati in un file libreria .h

Per esempio in un file del tipo :

```
#include "3DObjects.h"
```

Incluso all'inizio del codice.

Se la mesh non ha uno scheletro ed è semplicemente statica, questi semplici array contengono tutte le informazioni necessarie per disegnare la mesh correttamente.
Questi array vengono creati automaticamente con un programma di mia creazione che prende in input alcuni file di testo opportunamente formattati. La semplice operazione che bisognerà fare sarà un copia e incolla del file in output all'interno del famoso file .h.
C'è da notare che I tre file presi in input dal programma di mia creazione non sono altro che dovuti ad un altra semplice operazione di copia e incolla dal file in formato MilkShape3D ASCII. E' da notare che si assume che si usi una mesh singola e non tante mesh separate è quindi importante avere all'interno del file mesh un'unica mesh poligonale.

All'interno del primo file "vertices.txt" bisogna copiare tutti i dati dei vertici come in questo esempio qui sotto:

```
// MilkShape 3D ASCII

Frames: 30
Frame: 1

Meshes: 1
"Pikachu" 0 0
513
0 -0.394134 -4.031174 -56.168152 0.738136 0.872696 5
0 -0.339256 -7.681541 -74.948296 0.751427 0.888163 5
…….
0 0.606553 0.814919 -99.120064 0.756648 0.852161 5
0 28.248407 44.688095 -5.897087 0.453614 0.806055 2
389
0.994053 -0.106262 0.023818
0.988495 -0.147908 0.031637
0.992406 -0.119851 0.027690
……


// vertices.txt file
```

```
0 -0.394134 -4.031174 -56.168152 0.738136 0.872696 5
0 -0.339256 -7.681541 -74.948296 0.751427 0.888163 5
……..
0 0.606553 0.814919 -99.120064 0.756648 0.852161 5
0 28.248407 44.688095 -5.897087 0.453614 0.806055 2
```

E' DA NOTARE CHE BISOGNA TENERE IN CONSIDERAZIONE SOLO I NUMERI COMPRESI TRA IL NUMERO DI VERTICI E QUELLO DELLE NORMALI (in questo esempio tra 513 e 389). NESSUN ALTRO NUMERO DEVE COMPARIRE ALL'INTERNO DEL FILE, CHE DEVE TERMINARE CON UNA LINEA VUOTA O A CAPO.

Nel secondo file "normals.txt" bisogna copiare solo i dati relativi alle normali come in questo esempio:

```
// MilkShape 3D ASCII

Frames: 30
Frame: 1

Meshes: 1
"Pikachu" 0 0
513
0 -0.394134 -4.031174 -56.168152 0.738136 0.872696 5
0 -0.339256 -7.681541 -74.948296 0.751427 0.888163 5
……
0 0.606553 0.814919 -99.120064 0.756648 0.852161 5
0 28.248407 44.688095 -5.897087 0.453614 0.806055 2
389
0.994053 -0.106262 0.023818
0.988495 -0.147908 0.031637
0.992406 -0.119851 0.027690
……
0.857360 0.227740 0.461593
-0.671372 -0.151214 -0.725530
-0.808161 -0.486752 -0.331584
0.799173 -0.534374 -0.275258
380
0 0 1 2 0 1 2 1
0 3 4 5 3 4 5 1
0 5 6 3 5 6 3 1
0 7 8 9 7 7 7 1
……
```

```
// normals.txt file

0.994053 -0.106262 0.023818
0.988495 -0.147908 0.031637
0.992406 -0.119851 0.027690
……
0.857360 0.227740 0.461593
-0.671372 -0.151214 -0.725530
-0.808161 -0.486752 -0.331584
0.799173 -0.534374 -0.275258
```

E' DA NOTARE CHE BISOGNA TENERE IN CONSIDERAZIONE SOLO I NUMERI COMPRESI TRA IL NUMERO DELLE NORMALI E QUELLO DELLE FACCE (in questo esempio tra 389 e 380). NESSUN ALTRO NUMERO DEVE COMPARIRE ALL'INTERNO DEL FILE, CHE DEVE TERMINARE CON UNA LINEA VUOTA O A CAPO.

Nell’ultimo file “faces.txt” bisogna copiare solo i dati relative alle informazioni sulle face come nell’esempio sottostante:

```
// MilkShape 3D ASCII

Frames: 30
Frame: 1

Meshes: 1
"Pikachu" 0 0
513
0 -0.394134 -4.031174 -56.168152 0.738136 0.872696 5
0 -0.339256 -7.681541 -74.948296 0.751427 0.888163 5
……
0 0.606553 0.814919 -99.120064 0.756648 0.852161 5
0 28.248407 44.688095 -5.897087 0.453614 0.806055 2
389
0.994053 -0.106262 0.023818
0.988495 -0.147908 0.031637
0.992406 -0.119851 0.027690
……
0.857360 0.227740 0.461593
-0.671372 -0.151214 -0.725530
-0.808161 -0.486752 -0.331584
0.799173 -0.534374 -0.275258
380
0 0 1 2 0 1 2 1
0 3 4 5 3 4 5 1
0 5 6 3 5 6 3 1
0 7 8 9 7 7 7 1
……
0 459 449 448 343 333 332 1
0 496 39 452 374 36 336 1
0 496 414 39 374 298 36 1

Materials: 2
"lambert1"
0.000000 0.000000 0.000000 1.000000
0.000000 0.000000 0.000000 1.000000
0.000000 0.000000 0.000000 1.000000
0.000000 0.000000 0.000000 1.000000
0.000000
1.000000
"pikachu.jpg"


// faces.txt file

0 0 1 2 0 1 2 1
0 3 4 5 3 4 5 1
0 5 6 3 5 6 3 1
0 7 8 9 7 7 7 1
……
0 459 449 448 343 333 332 1
0 496 39 452 374 36 336 1
0 496 414 39 374 298 36 1
```

E’ DA NOTARE CHE BISOGNA TENERE IN CONSIDERAZIONE SOLO I NUMERI COMPRESI TRA IL NUMERO DELLE FACCE E LA PARTE RELATIVA AI MATERIALI DELLA MESH (in questo esempio tra 380 e “Material: 2”). NESSUN ALTRO NUMERO DEVE COMPARIRE ALL’INTERNO DEL FILE, CHE DEVE TERMINARE CON UNA LINEA VUOTA O A CAPO.

Una volta riempiti I file l'unica operazione rimanente è il doppio click sul file MESH_DATA.bat e verrà creato un file "outputModel.txt" con il codice sorgente da copiare ed incollare nell'applicazione stessa.

Questo software MESH_DATA si può scaricare facilmente qui.
[http://www.gents.it/ggd/Files/MESH_DATA.zip]

Analizziamo ora come la funzione Load() lavora su questi dati per riempire gli array con dati utili alle OpenGL ES.
Questo codice sottostante assume un semplice implementazione di classi come Vertex e Face (non sono necessarie ai fini del funzionamento!!! Io le ho adoperate per dare maggior chiarezza):

```
// 23:9 fixed point math used trough all math in 3D-example
const TInt KShift = 9;
const TInt GlShift = 16;
const TInt shift = GlShift - KShift;

// CONSTANTS
const TUint KInvalidTextureObject = 0xffffffff;


void Mesh::Load(const TReal* verts, const TInt* faces, const TReal* textures,
                        const TReal* normals, TInt nv, TInt nf, TInt textureSize
)
{
      // filling mesh information data with arrays numbers
      iVertex = new TVertex[nv];
      iFace = new TFace[nf];

      num_vertices = nv;
      num_faces = nf;

      TInt i=0;
      TInt v=0, t=0, n=0;
      for(i=0; i<nv; i++)
      {
            iVertex[i] = TVertex( (TInt)verts[v]*100, (TInt)verts[v+1]*100,
(TInt)verts[v+2]*100,
                                          (TInt)textures[t],
(TInt)textures[t+1],
                                          (TInt)normals[n], (TInt)normals[n+1],
(TInt)normals[n+2] );
            v = v+3;
            t = t+2;
            n = n+3;
      }

      v=0;
      for(i=0; i<nf; i++)
      {
            iFace[i] = TFace( faces[v], faces[v+1], faces[v+2] );

            iFace[i].normalX = iVertex[iFace[i].iV1].normalX +
                                          iVertex[iFace[i].iV2].normalX +
                                          iVertex[iFace[i].iV3].normalX;
            iFace[i].normalY = iVertex[iFace[i].iV1].normalY +
                                          iVertex[iFace[i].iV2].normalY +
                                          iVertex[iFace[i].iV3].normalY;
            iFace[i].normalZ = iVertex[iFace[i].iV1].normalZ +
```

```
                                        iVertex[iFace[i].iV2].normalZ +
                                        iVertex[iFace[i].iV3].normalZ;
      v = v+3;
}

// delete OpenGL ES data and create new
delete[] iGLVerts;
delete[] iGLNormals;
delete[] iGLTexCoords;
delete[] iGLTris;

iGLTriCount = nf*3;
iGLVerts=new GLfixed[ iGLTriCount*3 ];
iGLTexCoords=new GLfixed[ iGLTriCount*2 ];
iGLTris=new GLushort[ iGLTriCount ];
iGLNormals=new GLfixed[ iGLTriCount*3 ];

TInt ic=0;
TInt vc=0;

// filling OpenGL ES with mesh information data
for (i=0;i<nf;i++)
{
      TInt a,b,c;
      a=iFace[i].iV1;
      b=iFace[i].iV2;
      c=iFace[i].iV3;

      iGLVerts[vc*3] = iVertex[a].iX << shift;
      iGLVerts[vc*3+1] = iVertex[a].iY << shift;
      iGLVerts[vc*3+2] = iVertex[a].iZ << shift;
            iGLNormals[vc*3] = iVertex[a].normalX << shift;
            iGLNormals[vc*3+1] = iVertex[a].normalY << shift;
            iGLNormals[vc*3+2]   = iVertex[a].normalZ << shift;
      iGLTexCoords[vc*2] = ( iVertex[a].iTx << GlShift ) / textureSize;
      iGLTexCoords[vc*2+1] = ( iVertex[a].iTy << GlShift ) / textureSize;
      iGLTris[ic] = (GLushort)vc;
      ic++;
      vc++;

      iGLVerts[vc*3] = iVertex[b].iX << shift;
      iGLVerts[vc*3+1] = iVertex[b].iY << shift;
      iGLVerts[vc*3+2] = iVertex[b].iZ << shift;
            iGLNormals[vc*3] = iVertex[b].normalX << shift;
            iGLNormals[vc*3+1] = iVertex[b].normalY << shift;
            iGLNormals[vc*3+2]   = iVertex[b].normalZ << shift;
      iGLTexCoords[vc*2] = ( iVertex[b].iTx << GlShift ) / textureSize;
      iGLTexCoords[vc*2+1] = ( iVertex[b].iTy << GlShift ) / textureSize;
      iGLTris[ic] = (GLushort)vc;
      ic++;
      vc++;

      iGLVerts[vc*3] = iVertex[c].iX << shift;
      iGLVerts[vc*3+1] = iVertex[c].iY << shift;
      iGLVerts[vc*3+2] = iVertex[c].iZ << shift;
            iGLNormals[vc*3] = iVertex[c].normalX << shift;
            iGLNormals[vc*3+1] = iVertex[c].normalY << shift;
            iGLNormals[vc*3+2]   = iVertex[c].normalZ << shift;
      iGLTexCoords[vc*2] = ( iVertex[c].iTx << GlShift) / textureSize;
      iGLTexCoords[vc*2+1] = ( iVertex[c].iTy << GlShift) / textureSize;
      iGLTris[ic] = (GLushort)vc;
      ic++;
      vc++;
```

```
	}

	// assign an invalid texture object
	iTextureObject = KInvalidTextureObject;

	// set a standard white material
	int l=0;
	for(l=0; l<4; l++)
	{
		matDiffuse[l]  = 1.0;
		matAmbient[l]  = 1.0;
		matSpecular[l] = 1.0;
		matEmission[l] = 1.0;
	}
	shininess = 0;

	visible = true;
}
```

Fatto ciò rimane solo da settare una texture per la mesh e poi disegnarla. Effettuare questa assegnazione è molto semplice. Ciò che appare più difficile è come caricare immagini bitmap o jpeg, ma questo non è parte di interesse in questo articolo. Si assumerà quindi la presenza di un oggetto del tipo GLuint iTexture:

```
void SetTexture( GLuint iTexture)
{
	iTextureObject = iTexture;
}
```

Ora si necessita solo di disegnare il tutto sullo schermo. Nella implementazione sottostante appaiono anche casi relativi alle mesh non statiche ma già con uno scheletro. Quello che dovrete fare ora è solo saltare queste parti per tornarci poi successivamente con calma quando avrò spiegato con più chiarezza il processo di caricamento delle informazioni per l'animazione.

```
void Mesh::Draw()
{

	TInt vc=0;

	// for all faces move and rotate vertices in 3D space
	for ( TInt j = 0; j < num_faces; j++ )
	{
		if ( skeletoned && iVertex[iFace[j].iV1].m_boneID >= 0 )
		{
			// rotate according to transformation matrix
			const Matrix& final =
m_pJoints[iVertex[iFace[j].iV1].m_boneID].m_final;

			TReal m_vertexNormals[3];
			m_vertexNormals[0] = iVertex[iFace[j].iV1].normalX;
			m_vertexNormals[1] = iVertex[iFace[j].iV1].normalY;
			m_vertexNormals[2] = iVertex[iFace[j].iV1].normalZ;

			Vector newNormal( m_vertexNormals );
			newNormal.transform3( final );
			iGLNormals[vc*3] = (TInt)newNormal.m_vector[0] << shift;
			iGLNormals[vc*3+1] = (TInt)newNormal.m_vector[1] << shift;
```

```
                iGLNormals[vc*3+2] = (TInt)newNormal.m_vector[2] << shift;

                TReal m_location[3];
                m_location[0] = iVertex[iFace[j].iV1].iX;
                m_location[1] = iVertex[iFace[j].iV1].iY;
                m_location[2] = iVertex[iFace[j].iV1].iZ;

                Vector newVertex( m_location );
                newVertex.transform( final );
                iGLVerts[vc*3] = (TInt)newVertex.m_vector[0] << shift;
                iGLVerts[vc*3+1] = (TInt)newVertex.m_vector[1] << shift;
                iGLVerts[vc*3+2] = (TInt)newVertex.m_vector[2] << shift;
            }
            else
            {
                iGLVerts[vc*3] = iVertex[iFace[j].iV1].iX << shift;
                iGLVerts[vc*3+1] = iVertex[iFace[j].iV1].iY << shift;
                iGLVerts[vc*3+2] = iVertex[iFace[j].iV1].iZ << shift;
                iGLNormals[vc*3] = iVertex[iFace[j].iV1].normalX << shift;
                iGLNormals[vc*3+1] = iVertex[iFace[j].iV1].normalY << shift;
                iGLNormals[vc*3+2]   = iVertex[iFace[j].iV1].normalZ << shift;
            }
            vc++;

            if ( skeletoned && iVertex[iFace[j].iV2].m_boneID >= 0 )
            {
                // rotate according to transformation matrix
                const Matrix& final =
m_pJoints[iVertex[iFace[j].iV2].m_boneID].m_final;

                TReal m_vertexNormals[3];
                m_vertexNormals[0] = iVertex[iFace[j].iV2].normalX;
                m_vertexNormals[1] = iVertex[iFace[j].iV2].normalY;
                m_vertexNormals[2] = iVertex[iFace[j].iV2].normalZ;

                Vector newNormal( m_vertexNormals );
                newNormal.transform3( final );
                iGLNormals[vc*3] = (TInt)newNormal.m_vector[0] << shift;
                iGLNormals[vc*3+1] = (TInt)newNormal.m_vector[1] << shift;
                iGLNormals[vc*3+2] = (TInt)newNormal.m_vector[2] << shift;

                TReal m_location[3];
                m_location[0] = iVertex[iFace[j].iV2].iX;
                m_location[1] = iVertex[iFace[j].iV2].iY;
                m_location[2] = iVertex[iFace[j].iV2].iZ;

                Vector newVertex( m_location );
                newVertex.transform( final );
                iGLVerts[vc*3] = (TInt)newVertex.m_vector[0] << shift;
                iGLVerts[vc*3+1] = (TInt)newVertex.m_vector[1] << shift;
                iGLVerts[vc*3+2] = (TInt)newVertex.m_vector[2] << shift;
            }
            else
            {
                iGLVerts[vc*3] = iVertex[iFace[j].iV2].iX << shift;
                iGLVerts[vc*3+1] = iVertex[iFace[j].iV2].iY << shift;
                iGLVerts[vc*3+2] = iVertex[iFace[j].iV2].iZ << shift;
                iGLNormals[vc*3] = iVertex[iFace[j].iV2].normalX << shift;
                iGLNormals[vc*3+1] = iVertex[iFace[j].iV2].normalY << shift;
                iGLNormals[vc*3+2]   = iVertex[iFace[j].iV2].normalZ << shift;
            }
            vc++;
```

```
            if ( skeletoned && iVertex[iFace[j].iV3].m_boneID >= 0 )
            {
                  // rotate according to transformation matrix
                  const Matrix& final =
m_pJoints[iVertex[iFace[j].iV3].m_boneID].m_final;

                  TReal m_vertexNormals[3];
                  m_vertexNormals[0] = iVertex[iFace[j].iV3].normalX;
                  m_vertexNormals[1] = iVertex[iFace[j].iV3].normalY;
                  m_vertexNormals[2] = iVertex[iFace[j].iV3].normalZ;

                  Vector newNormal( m_vertexNormals );
                  newNormal.transform3( final );
                  iGLNormals[vc*3] = (TInt)newNormal.m_vector[0] << shift;
                  iGLNormals[vc*3+1] = (TInt)newNormal.m_vector[1] << shift;
                  iGLNormals[vc*3+2] = (TInt)newNormal.m_vector[2] << shift;

                  TReal m_location[3];
                  m_location[0] = iVertex[iFace[j].iV3].iX;
                  m_location[1] = iVertex[iFace[j].iV3].iY;
                  m_location[2] = iVertex[iFace[j].iV3].iZ;

                  Vector newVertex( m_location );
                  newVertex.transform( final );
                  iGLVerts[vc*3] = (TInt)newVertex.m_vector[0] << shift;
                  iGLVerts[vc*3+1] = (TInt)newVertex.m_vector[1] << shift;
                  iGLVerts[vc*3+2] = (TInt)newVertex.m_vector[2] << shift;
            }
            else
            {
                  iGLVerts[vc*3] = iVertex[iFace[j].iV3].iX << shift;
                  iGLVerts[vc*3+1] = iVertex[iFace[j].iV3].iY << shift;
                  iGLVerts[vc*3+2] = iVertex[iFace[j].iV3].iZ << shift;
                  iGLNormals[vc*3] = iVertex[iFace[j].iV3].normalX << shift;
                  iGLNormals[vc*3+1] = iVertex[iFace[j].iV3].normalY << shift;
                  iGLNormals[vc*3+2]   = iVertex[iFace[j].iV3].normalZ << shift;
            }
            vc++;
      }

      // begin OpenGL ES drawing phase

      glMatrixMode(GL_MODELVIEW);

      glEnableClientState( GL_VERTEX_ARRAY );
      glVertexPointer( 3, GL_FIXED, 0, iGLVerts );

      glEnableClientState( GL_NORMAL_ARRAY );
      glNormalPointer(  GL_FIXED, 0, iGLNormals );

      // check if the mesh is textured
      if( iTextureObject != KInvalidTextureObject )
      {
            glBindTexture( GL_TEXTURE_2D, iTextureObject );
            glEnable( GL_TEXTURE_2D );
            glEnableClientState( GL_TEXTURE_COORD_ARRAY );
            glTexCoordPointer( 2, GL_FIXED, 0, iGLTexCoords );
      }
      else
      {
            glDisable( GL_TEXTURE_2D );
            glDisableClientState ( GL_TEXTURE_COORD_ARRAY );
      }
```

```
      // Set mesh material
      glMaterialfv(   GL_FRONT_AND_BACK, GL_DIFFUSE,  matDiffuse  );
      glMaterialfv(   GL_FRONT_AND_BACK, GL_AMBIENT,  matAmbient  );
      glMaterialfv(   GL_FRONT_AND_BACK, GL_SPECULAR, matSpecular );
      glMaterialfv(   GL_FRONT_AND_BACK, GL_EMISSION, matEmission );
      glMaterialx(      GL_FRONT_AND_BACK, GL_SHININESS,   shininess << 16    );

      // draw all mesh data
      glColor4x( 255.f, 255.f, 255.f, 255.f ); //White
      glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST );
      glDrawElements( GL_TRIANGLES, iGLTriCount, GL_UNSIGNED_SHORT, iGLTris );

      glDisableClientState( GL_VERTEX_ARRAY );
      glDisableClientState( GL_NORMAL_ARRAY );

}
```

Ora possiamo usare tutti i metodi sopra descritti per caricare e disegnare la mesh sullo schermo come in questo esempio:

```
    // Set the screen background color.
    glClearColor( 1.0f, 1.0f, 1.0f, 1.0f );

    /* Make OpenGL ES automatically normalize all normals after tranformations.
       This is important when making irregular xforms like scaling, or if we
       have specified nonunit-length normals. */
    glEnable( GL_NORMALIZE  );

    // Initialize viewport and projection.
    glViewport( 0, 0, iScreenWidth, iScreenHeight );
    glMatrixMode( GL_PROJECTION );
    glFrustumf( -1.f, 1.f, -1.f, 1.f, 3.f, 1000.f );

    glMatrixMode( GL_MODELVIEW );

      Ambient.Load( KAmbientVertexData,   KAmbientFaceData,
                        KAmbientTextureData, KAmbientNormalData,
                        KNumAmbientVertices, KNumAmbientFaces, 256 );

      Ambient.SetAmbientMaterial( 0.8, 0.8, 0.8, 1.0 );
      Ambient.SetDiffuseMaterial( 0.8, 0.8, 0.8, 1.0 );
      Ambient.SetEmissionMaterial( 0.8, 0.8, 0.8, 1.0 );
      Ambient.SetSpecularMaterial( 0.8, 0.2, 0.4, 1.0 );
      Ambient.SetShininess( 1 );

      Ambient.SetTexture( iTexObjects[1] );

      glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

      glShadeModel( GL_SMOOTH );

      // Set the screen background color.
      glClearColor( 0.0f, 0.0f, 0.0f, 1.0f );

      glLoadIdentity();

      //Draw
      glPushMatrix();

      Ambient.SetPosition( 0, 0, -100 );
      Ambient.Scale( 15, 4, 15 );
      Ambient.Draw();
```

```
      glPopMatrix();
```
E questo è tutto per quanto riguarda le mesh statiche.
Ora immagino che starete dicendo qualcosa del tipo: “vorrà farci sapere qualcosa sulle mesh animate con scheletro, spero”. Beh… OK!!!

Per prima cosa bisogna ottenere in qualche modo le informazioni riguardanti ossa e keyframes dal file in formato MilkShape3D ASCII. Ma anche questa operazione apparirà molto semplice in quanto ho creato un altro piccolo programma che renderà tutto semplice ed intuitivo, per permettere ancora una volta di effettuare un veloce copia e incolla nel file .h.
Basterà seguire delle semplici istruzioni come nell’esempio qui sotto:

```
//MilkShape 3D ASCII

……
QUI CI SONO I DATI SUI MATERIALI
……

Bones: 10
"joint1"
""
16 -9.000000 -41.788368 -28.533302 0.000000 0.000000 0.000000
2
1.000000 0.000000 0.000000 0.000000
30.000000 0.000000 0.000000 0.000000
2
1.000000 0.000000 0.000000 0.000000
30.000000 0.000000 0.000000 0.000000
"joint2"
"joint1"
16 0.527245 21.499857 38.500114 0.000000 0.000000 0.000000
7
1.000000 0.000000 0.000000 0.000000
10.000000 0.000000 0.000000 0.000000
12.000000 0.000000 0.000000 0.000000
……
……
……
20.000000 -0.418879 0.000000 0.000000
30.000000 0.000000 0.000000 0.000000
GroupComments: 0
MaterialComments: 0



//joints.txt file

Bones: 10
"joint1"
""
16 -9.000000 -41.788368 -28.533302 0.000000 0.000000 0.000000
2
1.000000 0.000000 0.000000 0.000000
30.000000 0.000000 0.000000 0.000000
2
1.000000 0.000000 0.000000 0.000000
30.000000 0.000000 0.000000 0.000000
"joint2"
"joint1"
16 0.527245 21.499857 38.500114 0.000000 0.000000 0.000000
7
1.000000 0.000000 0.000000 0.000000
```

```
10.000000 0.000000 0.000000 0.000000
12.000000 0.000000 0.000000 0.000000
……
……
……
20.000000 -0.418879 0.000000 0.000000
30.000000 0.000000 0.000000 0.000000
```

E' DA NOTARE CHE BISOGNA TENERE IN CONSIDERAZIONE SOLO LE LINEE COMPRESE TRA IL NUMERO DI OSSA E LA PARTE RELATIVA ALLE ULTIME NOTE (in questo esempio tra "Bones: 10" e "GroupComments: 0"). NESSUN ALTRO NUMERO DEVE COMPARIRE ALL'INTERNO DEL FILE, CHE DEVE TERMINARE CON UNA LINEA VUOTA O A CAPO.

Una volt ache il file "joints.txt" sarà stato riempito, basterà effettuare un doppio click sull'eseguibile SKELETON_DATA.bat e due file ("JointsData.txt" e "KeyframesData.txt") verranno create con il codice sorgente da includere direttamente nell'applicazione.

Questo software SKELETON_DATA si può scaricare facilmente qui.
[http://www.gents.it/ggd/Files/SKELETON_DATA.zip]

Analizziamo ora come la funzione LoadAnimationData() lavora con questi dati direttamente.
Questo codice, sotto presentato, presuppone l'utilizzo di specifiche classi come Matrix, Quaternion e Vector (maggiori informazioni si possono trovare a questo indirizzo http://rsn.gamedev.net/tutorials/ms3danim.asp) :

```
void Mesh::LoadAnimationData( TInt totTime, TInt nJoints, const TReal*
jointData, const TReal* keyframeData, const TInt* vertexJointData )
{
	// Let us know that we want to draw a skinned mesh
	skeletoned = true;

	// set total length of the animation
	m_totalTime = totTime*1000;

	// set number of joints
	m_numJoints = nJoints;

	m_pJoints = new Joint[m_numJoints];

	TInt i = 0;

	// fill vertices structure with bone associated
	for(i=0; i<num_vertices; i++)
		iVertex[i].m_boneID = vertexJointData[i];

	TInt j=0;
	TInt k=0;

	// for every joint fill keyframe data
	for ( i = 0; i < nJoints; i++ )
	{
			j++;
			m_pJoints[i].m_parent = (TInt)jointData[j];
			j++;
			m_pJoints[i].m_localTranslation[0] = (TInt)jointData[j] <<
shift;
			j++;
```

```
                  m_pJoints[i].m_localTranslation[1] = (TInt)jointData[j] <<
shift;
                  j++;
                  m_pJoints[i].m_localTranslation[2] = (TInt)jointData[j] <<
shift;
                  j++;
                  m_pJoints[i].m_localRotation[0] = jointData[j];
                  j++;
                  m_pJoints[i].m_localRotation[1] = jointData[j];
                  j++;
                  m_pJoints[i].m_localRotation[2] = jointData[j];
                  j++;


                        k++;
                        TInt l = 0;

                        m_pJoints[i].m_numTranslationKeyframes =
(TInt)keyframeData[k];
                        m_pJoints[i].m_pTranslationKeyframes = new
Keyframe[(TInt)keyframeData[k]];
                        k++;

                        for ( l = 0; l < m_pJoints[i].m_numTranslationKeyframes;
l++ )
                        {
                              TInt time = keyframeData[k];
                              k++;
                              TReal m_parameter[3];
                              m_parameter[0] = (TInt)keyframeData[k] << shift;
                              k++;
                              m_parameter[1] = (TInt)keyframeData[k] << shift;
                              k++;
                              m_parameter[2] = (TInt)keyframeData[k] << shift;
                              k++;
                              setJointKeyframe( i, l, time*1000, m_parameter,
false );
                        }

                        m_pJoints[i].m_numRotationKeyframes =
(TInt)keyframeData[k];
                        m_pJoints[i].m_pRotationKeyframes = new
Keyframe[(TInt)keyframeData[k]];
                        k++;

                        for ( l = 0; l < m_pJoints[i].m_numRotationKeyframes;
l++ )
                        {
                              TInt time = keyframeData[k];
                              k++;
                              TReal m_parameter[3];
                              m_parameter[0] = keyframeData[k];
                              k++;
                              m_parameter[1] = keyframeData[k];
                              k++;
                              m_parameter[2] = keyframeData[k];
                              k++;
                              setJointKeyframe( i, l, time*1000, m_parameter,
true );
                        }

      }
```

```
	setupJoints();

	animActive = 0;
	nothingToDo = false;
	restart(animActive);

	jointData = NULL;
	keyframeData = NULL;
	vertexJointData = NULL;

}
```

Potete notare che questo metodo sopra descritto fa uso di alcune funzioni. Mostrerò di seguito il loro codice:

```
void Mesh::setJointKeyframe( TInt jointIndex, TInt keyframeIndex, TInt time,
TReal *parameter, bool isRotation )
{
	// fill keyframe data for a single joint with some parameter passed in
input at this function
	Keyframe& keyframe = isRotation ?
m_pJoints[jointIndex].m_pRotationKeyframes[keyframeIndex] :
		m_pJoints[jointIndex].m_pTranslationKeyframes[keyframeIndex];

	keyframe.m_jointIndex = jointIndex;
	keyframe.m_time = time;
	keyframe.m_parameter[0] = parameter[0];
	keyframe.m_parameter[1] = parameter[1];
	keyframe.m_parameter[2] = parameter[2];
}



void Mesh::setupJoints()
{

	// this function set up every joint to their initial position so to have
all dependencies working well
	TInt i=0;
	for ( i = 0; i < m_numJoints; i++ )
	{
		Joint& joint = m_pJoints[i];

		// we set rotation and translation for every koint
		joint.m_relative.setRotationRadians( joint.m_localRotation );
		joint.m_relative.setTranslation( joint.m_localTranslation );
		// and modify them if it's not a leaf node
		if ( joint.m_parent >= 0 )
		{
			joint.m_absolute.set(
m_pJoints[joint.m_parent].m_absolute.m_matrix );
			joint.m_absolute.postMultiply( joint.m_relative );
		}
		else
			joint.m_absolute.set( joint.m_relative.m_matrix );
	}

	// move every vertices accordingly to koint transformations
	for ( i = 0; i < num_vertices; i++ )
	{
		if ( iVertex[i].m_boneID >= 0 )
		{
```

```
                const Matrix& matrix =
m_pJoints[iVertex[i].m_boneID].m_absolute;

                TReal m_location[3];
                m_location[0] = iVertex[i].iX;
                m_location[1] = iVertex[i].iY;
                m_location[2] = iVertex[i].iZ;
                matrix.inverseTranslateVect( m_location );
                matrix.inverseRotateVect( m_location );
                iVertex[i].iX = (TInt)m_location[0];
                iVertex[i].iY = (TInt)m_location[1];
                iVertex[i].iZ = (TInt)m_location[2];

                TReal m_vertexNormals[3];
                m_vertexNormals[0] = iVertex[i].normalX;
                m_vertexNormals[1] = iVertex[i].normalY;
                m_vertexNormals[2] = iVertex[i].normalZ;
                matrix.inverseRotateVect( m_vertexNormals );
                iVertex[i].normalX = (TInt)m_vertexNormals[0];
                iVertex[i].normalY = (TInt)m_vertexNormals[1];
                iVertex[i].normalZ = (TInt)m_vertexNormals[2];
            }
        }
}


// This function let us to set the animation number n from the beginning. It
supposes that we've initialized the AnimationSet array of Mesh class in another
part of our code
void Mesh::restart(int n)
{
        for ( TInt i = 0; i < m_numJoints; i++ )
        {
            m_pJoints[i].m_currentRotationKeyframe =
m_pJoints[i].m_currentTranslationKeyframe = 0;
            m_pJoints[i].m_final.set( m_pJoints[i].m_absolute.getMatrix());
        }

        if(n==0)
            m_Time = 0;
        else
            m_Time = animation[n].initTime*1000;

        nothingToDo = false;
}

// This function works almost as restart method but in this we set animActive
variable
void Mesh::setAnimationSet(int n)
{
        animActive = n;

        for ( TInt i = 0; i < m_numJoints; i++ )
        {
            m_pJoints[i].m_currentRotationKeyframe =
m_pJoints[i].m_currentTranslationKeyframe = 0;
            m_pJoints[i].m_final.set( m_pJoints[i].m_absolute.getMatrix());
        }

        m_Time = animation[n].initTime*1000;
        nothingToDo = false;
}
```

```
// This function is the core of all animation. We have to call this method befor
drawing our mesh or it'll seem us static
void Mesh::advanceAnimation()
{
	// let time pass
	m_Time += 1000;

	TInt time = m_Time;

	// Here we check if we have to restart our animation if it haven't to loop
	if ( time > animation[animActive].endTime*1000 )
	{
		if ( m_looping )
		{
			restart(animActive);
			time = animation[animActive].initTime*1000;
		}
		else
		{
			time = animation[animActive].endTime*1000;
			nothingToDo = true;
		}
	}

	// Now we set all transformation following keyframe transformation
parameters
	for ( TInt i = 0; i < m_numJoints; i++ )
	{
		TReal transVec[3];
		Matrix transform;
		TInt frame;
		Joint *pJoint = &m_pJoints[i];

		if ( pJoint->m_numRotationKeyframes == 0 && pJoint-
>m_numTranslationKeyframes == 0 )
		{
			pJoint->m_final.set( pJoint->m_absolute.getMatrix());
			continue;
		}

		frame = pJoint->m_currentTranslationKeyframe;
		while ( frame < pJoint->m_numTranslationKeyframes && pJoint-
>m_pTranslationKeyframes[frame].m_time < time )
		{
			frame++;
		}
		pJoint->m_currentTranslationKeyframe = frame;

		if ( frame == 0 ){
			transVec[0] = pJoint-
>m_pTranslationKeyframes[0].m_parameter[0];
			transVec[1] = pJoint-
>m_pTranslationKeyframes[0].m_parameter[1];
			transVec[2] = pJoint-
>m_pTranslationKeyframes[0].m_parameter[2];
		}
		else if ( frame == pJoint->m_numTranslationKeyframes ){
			transVec[0] = pJoint->m_pTranslationKeyframes[frame-
1].m_parameter[0];
			transVec[1] = pJoint->m_pTranslationKeyframes[frame-
1].m_parameter[1];
```

```
                transVec[2] = pJoint->m_pTranslationKeyframes[frame-
1].m_parameter[2];
            }
            else
            {
                const Mesh::Keyframe& curFrame = pJoint-
>m_pTranslationKeyframes[frame];
                const Mesh::Keyframe& prevFrame = pJoint-
>m_pTranslationKeyframes[frame-1];

                TReal timeDelta = curFrame.m_time-prevFrame.m_time;
                TReal interpValue = ( TReal )(( time-prevFrame.m_time
)/timeDelta );

                transVec[0] = (prevFrame.m_parameter[0]+(
curFrame.m_parameter[0]-prevFrame.m_parameter[0] )*interpValue);
                transVec[1] = (prevFrame.m_parameter[1]+(
curFrame.m_parameter[1]-prevFrame.m_parameter[1] )*interpValue);
                transVec[2] = (prevFrame.m_parameter[2]+(
curFrame.m_parameter[2]-prevFrame.m_parameter[2] )*interpValue);
            }

            frame = pJoint->m_currentRotationKeyframe;
            while ( frame < pJoint->m_numRotationKeyframes && pJoint-
>m_pRotationKeyframes[frame].m_time < time )
            {
                frame++;
            }
            pJoint->m_currentRotationKeyframe = frame;

            if ( frame == 0 )
                transform.setRotationRadians( pJoint-
>m_pRotationKeyframes[0].m_parameter );
            else if ( frame == pJoint->m_numRotationKeyframes )
                transform.setRotationRadians( pJoint-
>m_pRotationKeyframes[frame-1].m_parameter );
            else
            {
                const Mesh::Keyframe& curFrame = pJoint-
>m_pRotationKeyframes[frame];
                const Mesh::Keyframe& prevFrame = pJoint-
>m_pRotationKeyframes[frame-1];

                TReal timeDelta = curFrame.m_time-prevFrame.m_time;
                TReal interpValue = ( TReal )(( time-prevFrame.m_time
)/timeDelta );

                TReal rotVec[3];

                rotVec[0] = prevFrame.m_parameter[0]+(
curFrame.m_parameter[0]-prevFrame.m_parameter[0] )*interpValue;
                rotVec[1] = prevFrame.m_parameter[1]+(
curFrame.m_parameter[1]-prevFrame.m_parameter[1] )*interpValue;
                rotVec[2] = prevFrame.m_parameter[2]+(
curFrame.m_parameter[2]-prevFrame.m_parameter[2] )*interpValue;

                transform.setRotationRadians( rotVec );

            }

            transform.setTranslation( transVec );
            Matrix relativeFinal( pJoint->m_relative );
            relativeFinal.postMultiply( transform );
```

```
            if ( pJoint->m_parent < 0 )
                  pJoint->m_final.set( relativeFinal.getMatrix());
            else
            {
                  pJoint->m_final.set( m_pJoints[pJoint-
>m_parent].m_final.getMatrix());
                  pJoint->m_final.postMultiply( relativeFinal );
            }
      }
}
```

Se pensate che sia troppo complicato quanto visto fino ad ora vi stupirete per quanto semplice sia usare i metodi fino ad ora descritti:

```
Food.LoadAnimationData( 3, NumFoodBones, KFoodJointsData, KFoodKeyframeData,
KFoodVertexJointData );

Food.animation = new Mesh::AnimationSet[3];
Food.animation[0].initTime = 1;
Food.animation[0].endTime = 1;
Food.animation[1].initTime = 2;
Food.animation[1].endTime = 2;
Food.animation[2].initTime = 3;
Food.animation[2].endTime = 3;

Food.setAnimationSet( 0 );
Food.setLooping( false );
Food.visible = false;

Food.SetPosition( -32, -73, 24 );

Food.SetTexture( iTexObjects[12] );

glPushMatrix();

Food.SetPosition( Food.position[0], Food.position[1], Food.position[2] );
Food.advanceAnimation();
Food.Draw();

glPopMatrix();
```

Spero di essere stato chiaro ed esaudiente e che non ci siano particolari problemi. Nel caso dovessero essercene visitate pure il mio sito web http://www.gents.it/ggd/ o scrivetemi le vostre perplessitò a questo indirizzo e-mail gents@email.it

Potrete trovare del codice sorgente qui.
[http://www.gents.it/ggd/Files/SOURCE_CODE.zip]

Ciauz,

Mauro Gentile

alias

GENTS.