# Cargar .ASE's

---

**Vertices.h**

```
#pragma once
#include <stdlib.h>

struct Vert3d {
	float	VX;
	float	VY;
	float	VZ;
	int			ID;
};

class CVertices
{
public:
	CVertices(int Num);
	~CVertices(void);
	void Anadir(float x, float y, float z, int id);
	Vert3d Devolver(int NumVer);
private:
	Vert3d	*Vert;
	int			NumVert;
};
```

---

**Vertices.cpp**

```
#include "vertices.h"

CVertices::CVertices(int Num)
{
	NumVert=0;
	Vert = new Vert3d[Num];
}

CVertices::~CVertices(void)
{
	delete [] Vert;
}

void CVertices::Anadir(float x, float y, float z, int id)
{
	Vert[NumVert].VX=x;
	Vert[NumVert].VY=y;
	Vert[NumVert].VZ=z;
	Vert[NumVert].ID=id;
	NumVert++;
}

Vert3d CVertices::Devolver (int NumVer)
{
	return this->Vert[NumVer];
}
```

---

**CargaASE.h**

```
#pragma once
#include <stdio.h>
#include <string.h>
#include <windows.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include "Vertices.h"
#pragma comment(lib, "opengl32.lib")
#pragma comment(lib, "glu32.lib")

struct SCara {
	int			VerticeA;
	int			VerticeB;
	int			VerticeC;
};

class CCargaASE
{
public:
	CCargaASE(char *fichero, unsigned int *Tex, int *NumTex, bool ConTex, bool ConNor, int IDobj);
	~CCargaASE(void);
	void MuestraASE();
	void DiNumObj(HWND hwnd);
private:
```

```
	int				ID;
	int				NumVert;
	int				NumObj;
	bool		ConText;
	bool		ConNorm;
	CVertices	*Vertex;
	SCara		*Caras;
	CVertices	*TexVer;
	SCara		*TexCar;
	CVertices	*NorCar;
	void CreaListas(int Ini, int Fin, int Cara, unsigned int Texture, int TIni, int TFin, BOOL definitive);
};
```

---

**CargaASE.cpp**

```
#include "cargaase.h"

CCargaASE::CCargaASE(char *fichero, unsigned int *Tex, int *NumTex, bool ConTex, bool ConNor, int IDobj)
{
	FILE *Fic;
	char datos[255];
	int Func=0;
	bool sal;
	bool Tsal;
	float x, y, z;
	int Temp;
	int VertIni=0;
	int NumCaras=0;
	int NumTC=0;
	int id;
	int CaraAct;
	int VertFin=0;
	int NumTV=0;
	int TVertIni=0;
	int TVertFin=0;
	int TCarAct=0;
	int TextAct=0;
	int NCarAct=0;

	NumVert=0;
	ID = IDobj * 10000;
	NumObj=0;
	ConText=ConTex;
	ConNorm=ConNor;

	Fic = fopen (fichero, "r");

	while (!feof(Fic))
	{
		fscanf (Fic, "%s", &datos);
		Temp=0;
		if(!strcmp (datos,"*MESH_NUMVERTEX"))
			fscanf (Fic, "%d", &Temp);

		NumVert += Temp;
		Temp=0;
		if (!strcmp (datos,"*MESH_NUMFACES"))
			fscanf (Fic, "%d", &Temp);
	NumCaras += Temp;
		Temp=0;
		if (!strcmp (datos,"*MESH_NUMTVERTEX"))
			fscanf (Fic, "%d", &Temp);
	NumTV += Temp;
		Temp=0;
	}

	Vertex = new CVertices (NumVert);
	Caras = new SCara [NumCaras];

	if (ConText) {
		TexVer = new CVertices (NumTV);
		TexCar = new SCara [NumCaras];
	}

	if (ConNorm) {
		NorCar = new CVertices(NumCaras);
	}

	NumCaras=0;

	rewind (Fic);
	while (!feof(Fic))
	{
		fscanf (Fic, "%s", &datos);
```

```
if (!strcmp (datos,"*MESH_NUMFACES"))
{
	fscanf (Fic, "%d", &NumCaras);
}

if (!strcmp (datos,"*MESH_VERTEX_LIST"))
{
	sal=false;
	while (!sal)
	{
		fscanf (Fic, "%s", &datos);
		if (!strcmp (datos,"{")) Func++;
		if (!strcmp (datos,"}")) Func--;
		if (Func==0) sal=true;
		if (!strcmp (datos,"*MESH_VERTEX"))
		{
			fscanf (Fic, "%d", &Temp);
			id=Temp;
			fscanf (Fic, "%f %f %f", &x, &y,&z);
			Vertex->Anadir (x, y, z, id);
		}
	}
	NumObj++;
	VertFin = VertIni + Temp;
}

if (!strcmp (datos,"*MESH_FACE_LIST"))
{
	sal=false;
	CaraAct=0;
	while (!sal)
	{
		fscanf (Fic, "%s", &datos);
		if (!strcmp (datos,"{")) Func++;
		if (!strcmp (datos,"}")) Func--;
		if (Func==0) sal=true;
		if (!strcmp (datos,"*MESH_FACE"))
		{
			fscanf (Fic, "%s", &datos);
			fscanf (Fic, "%s", &datos);
			fscanf (Fic, "%d", &Caras[CaraAct].VerticeA);
			fscanf (Fic, "%s", &datos);
			fscanf (Fic, "%d", &Caras[CaraAct].VerticeB);
			fscanf (Fic, "%s", &datos);
			fscanf (Fic, "%d", &Caras[CaraAct].VerticeC);
			CaraAct++;
		}
	}


	if (ConText) {
		Tsal=false;
		while (!Tsal) {
			fscanf (Fic, "%s", &datos);
			if (!strcmp (datos,"*MESH_TVERTLIST"))
			{
				sal=false;
				while (!sal)
				{
					fscanf (Fic, "%s", &datos);
					if (!strcmp (datos,"{")) Func++;
					if (!strcmp (datos,"}")) Func--;
					if (Func==0) sal=true;
					if (!strcmp (datos,"*MESH_TVERT"))
					{
						fscanf (Fic, "%d", &Temp);
						id=Temp;
						fscanf (Fic, "%f %f %f", &x,
&y,&z);
						TexVer->Anadir (x, y, z, id);
					}
				}
				TVertFin = TVertIni + Temp;
			}

			if (!strcmp (datos,"*MESH_TFACELIST"))
			{
				sal=false;
				TCarAct=0;
				while (!sal)
				{
					fscanf (Fic, "%s", &datos);
					if (!strcmp (datos,"{")) Func++;
					if (!strcmp (datos,"}")) Func--;
					if (Func==0) sal=true;
					if (!strcmp (datos,"*MESH_TFACE"))
```

```
                                                                {
                                                                        fscanf (Fic, "%d", &TexCar[TCarAct]);
                                                                        fscanf (Fic, "%d %d %d",
&TexCar[TCarAct].VerticeA, &TexCar[TCarAct].VerticeB, &TexCar[TCarAct].VerticeC);
                                                                        TCarAct++;
                                                                }
                                                        }
                                                        Tsal=true;
                                                }
                                        }
                                }

                                Temp=NCarAct;
                                if (ConNorm) {
                                        sal=false;
                                        while (!sal)
                                        {
                                                fscanf (Fic, "%s", &datos);
                                                if (!strcmp (datos,"{")) Func++;
                                                if (!strcmp (datos,"}")) Func--;
                                                if (Func==0) sal=true;
                                                if (!strcmp (datos,"*MESH_FACENORMAL"))
                                                {
                                                        fscanf (Fic, "%d %f %f %f", &id, &x, &y,&z);
                                                        NorCar->Anadir (x,y,z,NCarAct);
                                                        NCarAct++;
                                                }
                                                if (NCarAct==Temp) sal=false;
                                        }
                                }

                                CreaListas(VertIni, VertFin, NumCaras, Tex[ NumTex[NumObj-1] ], TVertIni, TVertFin, false);
                                VertIni = VertFin+1;
                                TVertIni = TVertFin+1;
                        }
                }
                fclose (Fic);
                CreaListas(0,0,0,0,0,0,true);
}

CCargaASE::~CCargaASE(void)
{
}

void CCargaASE::CreaListas (int Ini, int Fin, int Cara, unsigned int Texture, int TIni, int TFin, BOOL definitive)
{
        int temp, i;
        temp=ID+NumObj;
        int a, b, c;

        if (definitive)
        {
                if (NumObj==1) ID=ID+1;
                else {
                        glNewList(ID, GL_COMPILE);
                                for (i=1; i<=NumObj; i++)
                                {
                                        a= ID + i;
                                        glCallList(a);
                                }
                        glEndList();
                }
        } else {
        glNewList(temp, GL_COMPILE);
                if (ConText)
                {
                        glBindTexture(GL_TEXTURE_2D, Texture);
                        glEnable(GL_TEXTURE_2D);
                }
                if (ConNorm)
                {

                }
                glBegin(GL_TRIANGLES);
                        for (i=0; i<=Cara; i++)
                        {
                                if (ConNorm)
                                {
                                        c=0;
                                        while (NorCar->Devolver (c).ID != c)
                                        {
                                                c++;
                                        }
                                        glNormal3f(NorCar->Devolver (c).VX, NorCar->Devolver (c).VY, NorCar-
>Devolver (c).VZ);
                                }
```

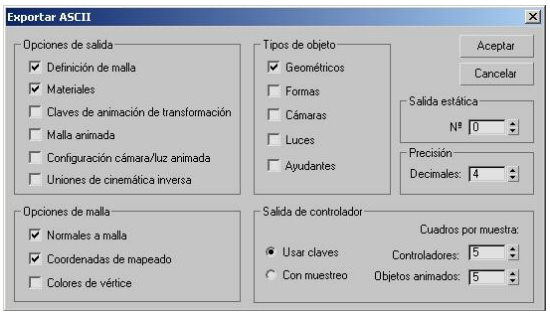
```
                                        for (a=Ini; a<=Fin; a++)
                                        {
                                                if (Caras[i].VerticeA == Vertex->Devolver (a).ID) break;
                                        }
                                                if (Caras[i].VerticeA == Vertex->Devolver (a).ID)
                                                {
                                                        if (ConText) {
                                                                for (b=TIni; b<=TFin; b++)
                                                                {
                                                                        if (TexCar[i].VerticeA == TexVer->Devolver
(b).ID) break;
                                                                }
                                                                if (TexCar[i].VerticeA == TexVer->Devolver (b).ID)
glTexCoord2f(TexVer->Devolver (b).VX, TexVer->Devolver (b).VY);
                                                        }
                                                        glVertex3f (Vertex->Devolver(a).VX, Vertex->Devolver(a).VZ,
Vertex->Devolver(a).VY);
                                                }

                                        for (a=Ini; a<=Fin; a++)
                                        {
                                                if (Caras[i].VerticeB == Vertex->Devolver (a).ID) break;
                                        }
                                                if (Caras[i].VerticeB == Vertex->Devolver (a).ID)
                                                {
                                                        if (ConText) {
                                                                for (b=TIni; b<=TFin; b++)
                                                                {
                                                                        if (TexCar[i].VerticeB == TexVer->Devolver
(b).ID) break;
                                                                }
                                                                if (TexCar[i].VerticeB == TexVer->Devolver (b).ID)
glTexCoord2f(TexVer->Devolver (b).VX, TexVer->Devolver (b).VY);
                                                        }
                                                        glVertex3f (Vertex->Devolver(a).VX, Vertex->Devolver(a).VZ,
Vertex->Devolver(a).VY);
                                                }

                                        for (a=Ini; a<=Fin; a++)
                                        {
                                                if (Caras[i].VerticeC == Vertex->Devolver (a).ID) break;
                                        }
                                                if (Caras[i].VerticeC == Vertex->Devolver (a).ID)
                                                {
                                                        if (ConText) {
                                                                for (b=TIni; b<=TFin; b++)
                                                                {
                                                                        if (TexCar[i].VerticeC == TexVer->Devolver
(b).ID) break;
                                                                }
                                                                if (TexCar[i].VerticeC == TexVer->Devolver (b).ID)
glTexCoord2f(TexVer->Devolver (b).VX, TexVer->Devolver (b).VY);
                                                        }
                                                        glVertex3f (Vertex->Devolver(a).VX, Vertex->Devolver(a).VZ,
Vertex->Devolver(a).VY);
                                                }
                                        }
                        glEnd();
                        if (ConText) glDisable(GL_TEXTURE_2D);
                glEndList();
                }
}

void CCargaASE::MuestraASE ()
{
        glCallList(ID);
}

void CCargaASE::DiNumObj (HWND hwnd)
{
        char Num[30];
        sprintf(Num, "Numero de objetos: %d", NumObj);
        MessageBox(hwnd, Num, "Objetos", NULL);
}
```

**Explicación:**

**Clase vértices:**
El constructor de esta clase, al que se le pasa un número, inicia un array de vértices con ese número. Y pone el contador del vertice actual a 0.
Existe una función miembro a la que se le pasan las coordenadas x, y, z del vertice y un identificador. Esta función incrementa el número del vertice actual.
La función Devolver devuelve el vertice pedido.

**Clase CargaASE**
Se basa en un array de vertices y en un array de una estructura llamada 'caras'. Una cara consta de 3 vertices. Esta clase tiene tres funciones importantes, la de crear el objeto ASE, la de dibujarlo y una privada en la que coloca todos los vertices, calcula las normales y hace todos los cálculos internos.
También disponemos de una función que calcula cuantos vertices tiene la figura y los muestra en un messagebox, más útil al programador que al usuario.

Los parámetros que se le pasan al constructor de esta clase son:
- El nombre del fichero (entre comillas).
- El array de texturas.
- Un vector con el orden de las texturas que aplicaremos a los objetos (si hay varios objetos).
- True o false según si los objetos están texturados o no.
- True o false si se calcularán las normales para los objetos.
- Un identificador para el ASE.




Estos son las opciones para exportar un ASE:
- Con definición de textura.
- Con materiales.
- Sólo los geométricos.
- Normales de la malla.
- Coordenadas de mapeado.

Como funciona:
- Indica a sus variables miembro si el programador permitirá las texturas y las normales en el objeto.
- Arregla el número del identificador para que el usuario no pueda repetirlo, ya que será el número de la display list que usará el objeto entero (si está formado por más objetos creará más display lists y luego las cargará en la del número del identificador).
- Abre el fichero y mira cuantos vertices, normales y caras tiene el objeto para crear los arrays.
- Luego irá buscando dentro de la estructura del ASE los vertices y lo colocará en su objeto de vertices, las texturas, las normales y las caras.
- Llamará a la función que crea las listas de visualización para cada objeto del ase, y cuando llegue al final del fichero creará la lista que englobará a las demás.
- La última función, lo único que hace es recorrer los arrays de vertices, caras, texturas y normales y llama a las funciones que dibujan un vertice (el orden lo indica la cara), posicionan la textura para este y llama a las normales.
- Una vez se creen todas las listas sólo habrá que llamarlas para que se ejecuten estos comandos y podamos ver el objeto por pantalla.

*Definición de objeto ASE y texturas*
```
CCargaASE *obj;
TGAFILE Img[2];
unsigned int texture[2];
```

*Carga de texturas*
```
LoadTGAFile ("Modelos/texture.tga", &Img[0]);
LoadTGAFile ("Modelos/texture2.tga", &Img[1]);

glGenTextures(2, texture);
glBindTexture(GL_TEXTURE_2D, texture[0]);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, Img[0].imageWidth , Img[0].imageHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, Img[0].imageData);

glBindTexture(GL_TEXTURE_2D, texture[1]);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, Img[1].imageWidth , Img[1].imageHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, Img[1].imageData);
```

*Definición del orden de texturas*
```
int OrdnTex[6] = {1,1,0,1,0,1};
```

*Crea el objeto*
```
obj = new CCargaASE("Modelos/statue.ase", texture, OrdnTex, true, true, 1);
```

*Muestra el objeto:*
```
obj->MuestraASE ();
```