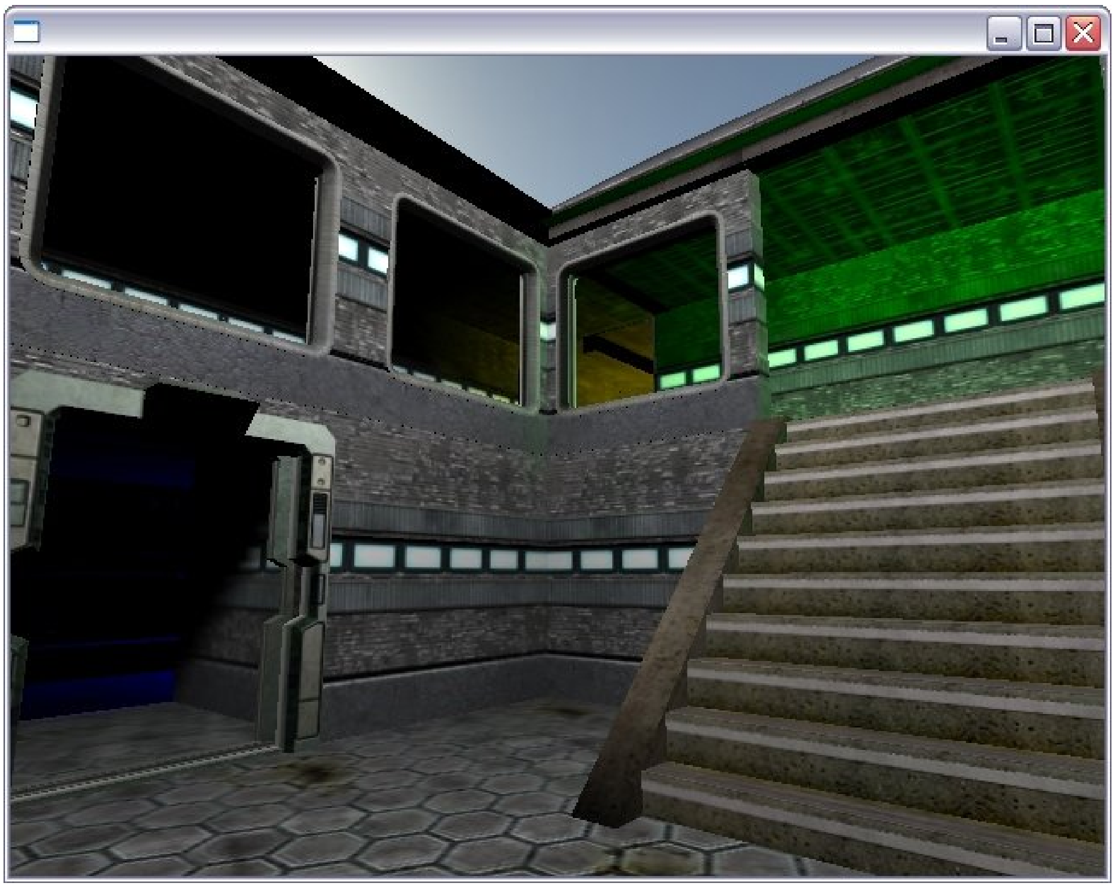
Loading 3D World Geometry

Introduction
This tutorial will guide you through the process of taking a game level, loading it and then being able to move around it.

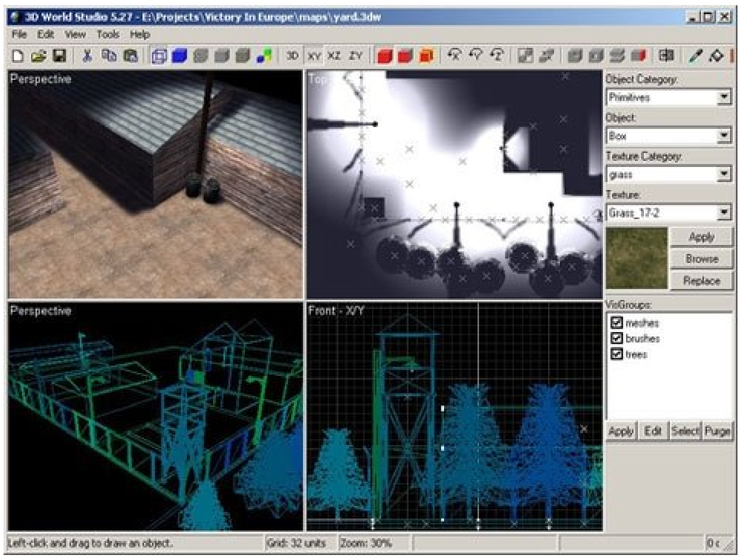


Overview of making game levels
While it is possible to make a game level or world using functions contained within Dark GDK (for example, using primitive shapes like boxes, spheres, cylinders etc), it is not really suitable, at least not for a more complex game. In situations where you want to create a large, detailed world it is necessary to turn to other software. Fortunately many options are available to you in the form of world editors and game maker software.

© 2007 The Game Creators Ltd. All Rights Reserved.

3D World Studio

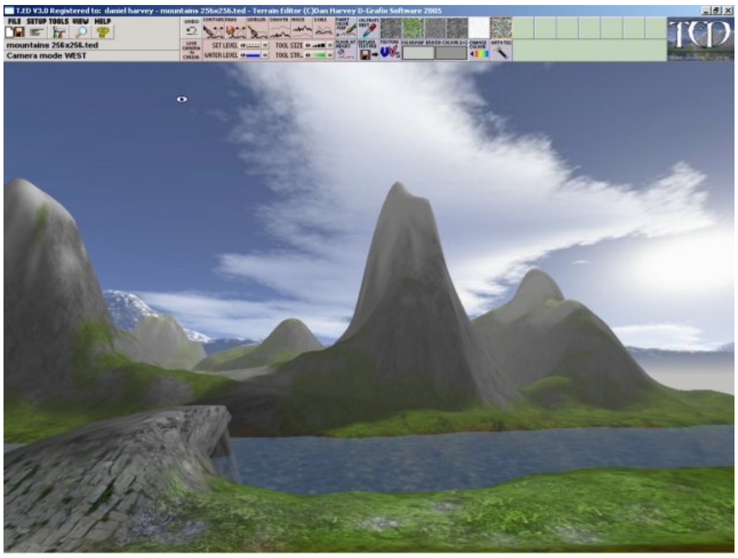
3D World Studio offers a feature set just as powerful as UnrealEd, yet it's friendly enough for beginners to use. New users can often start building in the program without reading a manual. It can export directly to formats supported in Dark GDK and is a great choice for creating game environments.



For more details on 3D World Studio and to download a demo visit:
http://3dworldstudio.thegamecreators.com/

T.ED Professional Terrain and Environment Editor

T.Ed is a dedicated terrain tool to allow the creation of small to large mesh landscapes, either as a series of blocks, or as a single terrain. It allows for superb blended textures either by using vertex alpha or by rendering a 'Supertexture' for each terrain block. Through an easy to use interface, featuring mouse look and game walk controls, you will be walking around your virtual lanscape in minutes.



For more details on T.ED visit: http://www.thegamecreators.com/?f=ted

© 2007 The Game Creators Ltd. All Rights Reserved.

FPS Creator
FPS Creator provides an easy-to-use yet highly flexible editing environment. As its name implies this is a tool for creating action-packed FPS games with no programming or 3D modelling knowledge required.

Using an intuitive and visual Windows interface you literally paint your game world into the scene. A vast range of 3D elements are included allowing you to paint hallways, corridors, gantries, walls, doors, access tunnels, ceilings, lifts, transporters, stairs and more. Segments intelligently attach themselves to each other - paint two corridor pieces side-by-side and they'll snap together seamlessly. Switch to 3D mode and you drop in on your scene for pixel perfect placement of 3D entities. Place a light-switch on the wall and it'll intelligently control the dynamic lighting in the room.

This may seem an unusual choice for hooking up with Dark GDK as FPS Creator is a complete game making kit. However, one of the neat features of FPS Creator is the ability to save your map into the DBO (Dark Basic Object) format. This format can be loaded directly into Dark GDK. It is a great choice for making game levels as it is incredibly easy to use and capable of very effective results.

For more information on FPS Creator visit www.fpscreator.com and www.fpscreatorx10.com

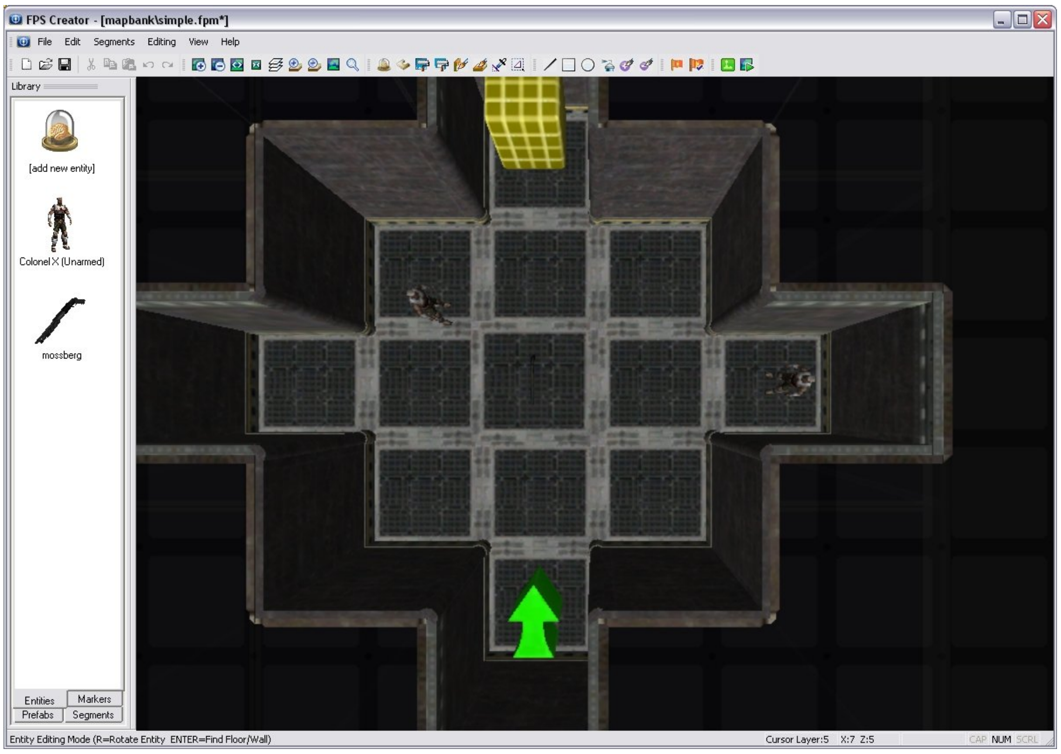
© 2007 The Game Creators Ltd. All Rights Reserved.

Making a level

For our tutorial the level will be made using FPS Creator. The main reasons for choosing FPS Creator are:

- We can make the level by placing simple blocks, it's very easy to do!
- It doesn't require any previous knowledge, you can be up and running within minutes.
- It's capable of creating excellent results.
- It's easy to use with Dark GDK

The FPS Creator editor looks like this:


FPS Creator - [mapbank\simple.fpm*]
File Edit Segments Editing View Help
Library
[add new entity]
Colonel X (Unarmed)
mossberg
Entities Markers
Prefabs Segments
Entity Editing Mode (R=Rotate Entity ENTER=Find Floor/Wall)
Cursor Layer:5 X:7 Z:5
CAP NUM SCRL

Once we have made our game level it is a simple case of exporting it and then copying the game files over to our project directory.

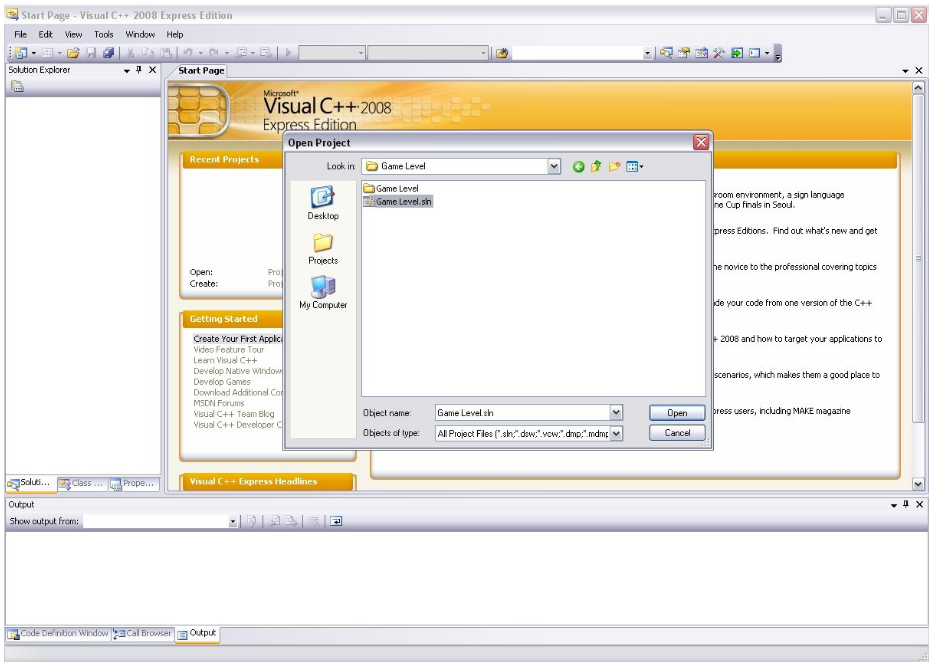
© 2007 The Game Creators Ltd. All Rights Reserved.

Opening the project

A project has already been created for this example. Launch Microsoft Visual C++ 2008 and go to the File menu and select Open and then Project. From here navigate to the directory where Dark GDK is installed. By default this is:

C:\Program Files\The Game Creators\Dark GDK

Now go into the Tutorials folder and then the Game Level folder. Finally open the solution file for Game Level.



Look in: Game Level

Game Level
Game Level.sln

Object name: Game Level.sln

Objects of type: All Project Files (*.sln;*.dsw;*.vcw;*.dmp;*.mdmp

Overview of code

The code for this project is relatively straight forward. It takes the following steps:

- Sets general properties such as sync rate
- Loads the game level and sets properties of it
- Creates a skybox
- Positions the camera
- Inside our main loop there is code to move the camera around

Initial setup

In the first few lines of our program there is a call to dbSyncOn and dbSyncRate:

```
dbSyncOn ( );
dbSyncRate ( 60 );
```

The first line will switch the sync rate on, meaning we are in control of when the screen is updated. In the second line we set the maximum refresh rate to 60 frames per second.

© 2007 The Game Creators Ltd. All Rights Reserved.

Loading our game level

Our game level has been created using FPS Creator and exported into the DBO format. This file can be loaded into Dark GDK by using the function dbLoadObject. This function takes two parameters, the first is a filename and the second is an ID number for the object. Here is the code for this section:

```
SetCurrentDirectory ( “media” );
dbLoadObject ( “universe.dbo”, 1 );
dbSetObjectLight ( 1, 0 );
```

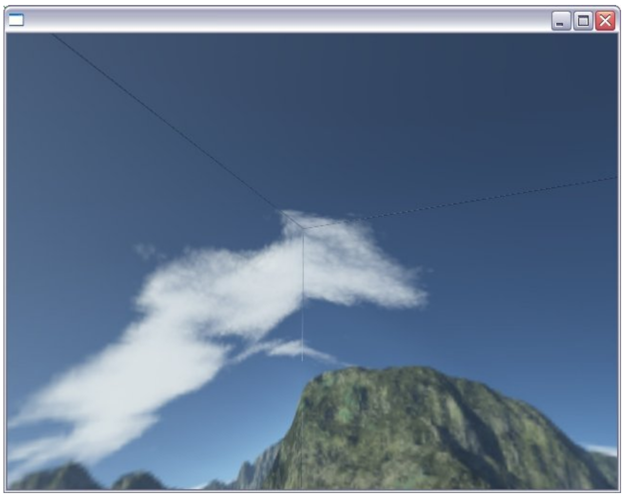
The first line will switch the current directory to the “media” folder. This folder contains all of our files for the game level. The function SetCurrentDirectory is part of the Win32 API. It is used to move the current directory which is by default the location where the executable is. The next line is a call to dbLoadObject. The filename of our model is passed in along with an ID number of 1. The third line switches lighting off for our object by calling dbSetObjectLight and passing in the ID number of our object and then 0 for the state. The reason for doing this is that our level has been lit using functionality contained in FPS Creator so it isn’t necessary for it to respond to light within our world.

Creating a skybox

The next block of code loads a model that will be used as our skybox. Once the model has been loaded lighting is switched off, it has a texture mode set and then it is scaled up so it will be larger than our level:

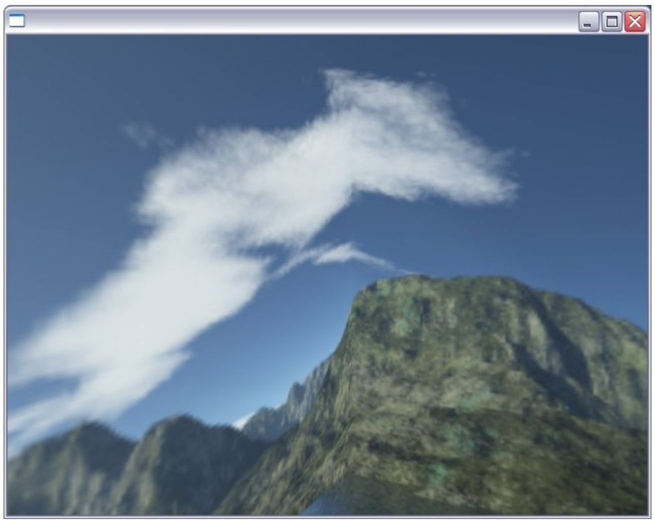
```
dbLoadObject ( “skybox2.x”, 2 );
dbSetObjectLight ( 2, 0 );
dbSetObjectTexture ( 2, 3, 2 );
dbScaleObject ( 2, 5000, 5000, 5000 );
```

By calling the function dbSetObjectTexture we can adjust some texture settings on our skybox. The reason behind this is that our skybox model has a slight problem. If you were to load the model, set the lighting off, scale up and miss out on this step then you would see the following:



© 2007 The Game Creators Ltd. All Rights Reserved.

Notice the lines along the edges of the skybox. This effect is happening due to the way the model has been created. This is easily solved by forcing the textures on the model to clamp at the edges by using dbSetObjectTexture, passing in our ID number for the object, then a value of 3 to set clamp mode on and finally a value of 2 to control the mipmapping state. By making this change we go from the above to the below:



By making this simple change in the code our skybox now looks correct and we have eliminated the seams.

Camera Setup
Within the next section of code the camera is positioned and two variables are declared:

```
dbPositionCamera ( 434, 42, -517 );

float fCameraAngleX = 0.0f;
float fCameraAngleY = 0.0f;
```

The two floating point variables will be used in the main loop in conjunction with other code to rotate the camera.

Main Loop
Our main loop starts off with a call to dbControlCameraUsingArrowKeys. This function will let the user move the camera forwards and backwards with the up and down arrow keys and turn left and right with the left and right arrow keys. It takes three parameters. The first parameter is an ID number. We have not created any new cameras and are therefore relying on the default camera so we pass in a value of 0. The second parameter controls how fast the camera moves when the up or down arrow key is pressed. If you are going to use this function then the movement speed is all dependant on the size of the world you have created. A value of 5.0 will allow us to move around our game level at a decent speed. Using a lower value will mean the camera moves slower while a higher value will increase the speed at which the camera moves. The final parameter controls how quickly the camera turns left or right when the left or right keys are pressed. A low value will result in the camera

© 2007 The Game Creators Ltd. All Rights Reserved.

turning slowly while a high value will increase the rate at which the camera turns. For our demo a value of 0.3 is used.

The next few lines of code are used so we can look around with the camera by using the mouse. What we want to achieve is the same kind of effect that you see in all first person shooter games. We want to rotate the camera and look around based on the movement of the mouse. Take a look at the following code:

```
fCameraAngleX = dbWrapValue ( fCameraAngleX + dbMouseMoveY ( )
* 0.4f );

fCameraAngleY = dbWrapValue ( fCameraAngleY + dbMouseMoveX ( )
* 0.4f );

dbXRotateCamera ( fCameraAngleX );
dbYRotateCamera ( fCameraAngleY );
```

The very first line deals with the camera angle on the X axis. It will create a new angle for the camera based on the current angle plus the movement by the mouse on the Y axis that is multiplied by 0.4. This is all then contained within a call to dbWrapValue which results in the angle being constrained within the range of 0 – 360. So when the mouse moves up or down it will control where the user wants to look and we can create a new rotation based on this movement. The extra step of multiplying the value by 0.4 is used as a method of controlling how fast the rotation happens. If for example, we didn't include this then when the user moves the mouse up or down then the new angle we create will move very quickly. This can be slowed by reducing the value with the multiplication. Later on when you run the program try adjusting this value for the X and Y axis and see how it affects the movement.

The second line will create a new rotation for the camera on the Y axis based on the movement of the mouse on the X axis. As you move your mouse to the left or right it will create movement and build an angle. This process is the same as that for the first line in which we set up a new angle for the X axis except this time we're basing movement on the X axis of the mouse.

The final two calls in this block of code simply rotate our camera on the X axis and then on our Y axis.

The last line of code in our program is a call to update the screen:

```
dbSync ( );
```

Conclusion

At this point our program is complete. Within a few lines we've been able to load a complex level, set up a skybox and be able to navigate around the level. As of yet there isn't any collision so you will be able to move the camera through walls etc. but you can see how easily the process is of loading a level into your world.

© 2007 The Game Creators Ltd. All Rights Reserved.