Using PagedGeometry
Tutorial 1: Getting Started

Introduction

Using PagedGeometry to display trees or grass is very easy, but it does require some configuration. It is important that you understand PagedGeometry's basic concepts (like PageLoader's and detail levels) in order to be able to setup LODs, etc. the way you want. While later tutorials will cover more practical applications of PagedGeometry, this tutorial is the most important to understand how PagedGeometry operates.

Tutorial Requirements

- ✔ A basic understanding of Ogre
- ✔ Some experience using C++

See also

- ◆ Example 1 - “TreeLoader3D”

Compiling PagedGeometry

The first thing you should do after extracting PagedGeometry somewhere is compile it. By default, the PagedGeometry library files (PagedGeometry.lib, and PagedGeometry_d.lib) are not included in the download, so you'll have to compile them yourself. Fortunately, this is fairly easy to do, since PagedGeometry doesn't require any external libraries besides Ogre. First, double-click on “PagedGeometry.sln” to open up the PagedGeometry library project in Visual Studio. Then find the “Solution Explorer” window right-click on “Solution 'PagedGeometry' (1 project)”, and select “Batch Build”. Click “Select All”, then “Build”, and wait until PagedGeometry has finished compiling (the status bar at the bottom of the screen should say “Build succeeded” when finished).

Next, you need to instruct your compiler that your project will be using PagedGeometry. Simply add PagedGeometry.lib in the “/lib” folder to your linker inputs, and make sure all the PagedGeometry header files are available to your compiler. If you're not sure how to do this, refer to the included examples.

As an alternate to the lib method described above, you can simply add all of PagedGeometry's .cpp files (found in the “/source” folder) to your project. Sometimes this is just as easy as the lib method.

Note: This tutorial assumes you already have Ogre up and running in your application, or at least have a basic understanding of what a Camera or an Entity is, for example. Complete executable examples of PagedGeometry can be found in the “/examples” folder under your PagedGeometry installation, if that's what you're looking for.

Creating PagedGeometry

Once PagedGeometry's classes and functions are available to your source code, you can start using it. First, create a new PagedGeometry object, like this:

```
PagedGeometry *trees = new PagedGeometry();
```

The PagedGeometry constructor allows two parameters to be supplied, but both are optional since they can be set later through some of PagedGeometry's functions. This tutorial will use the functions to configure these two options simply because it makes things a little more clear.

Note: What follows below describes how each option is set up and what it does. Remember that the order you call these functions makes no difference (with one excepition, which will be pointed out below), so don't worry about calling the functions in any certain order.

Usually the next thing you do after creating the PagedGeometry object is specify your camera:

```
trees->setCamera(camera);
```

Note that PagedGeometry was not designed for multiple cameras being rendered at the same time, although it's technically possible (setCamera() can be used to swap cameras just before each is rendered).

The camera is used to calculate LODs and cache pages of geometry smoothly. Without supplying a camera to PagedGeometry, nothing can be rendered since most of the internal optimization algorithms are based on the camera's position.

Now you need to give the PagedGeometry engine a little information about the size of your world, so it can optimize as best it can:

```
trees->setPageSize(50);
trees->setInfinite();
```

The two function calls above basically do two things:

1. setPageSize(50) sets the size of a single "page" of geometry to 50x50 units. Internally, everything is stored in a big grid of these blocks (called "pages"). Larger pages generally give better frame rates than smaller ones, although they can cause stuttering as they are dynamically loaded if too large. The page size is something that should be experimented with (later) to find the best performance.

2. setInfinite() instructs PagedGeometry not to force any boundaries on what geometry (trees, etc.) that you are going to add. "Infinite mode" is used by default, so normally you really don't need to call setInfinite() at all, but it is used here for demonstration.

As an alternate to setInfinite(), you could call PagedGeometry::setBounds(). This function allows you to supply PagedGeometry with the boundaries where all your trees (or anything else this PagedGeometry object will manage) will be contained. The only advantage to setBounds() over setInfinite() is that you may get a small speed boost, since the cache efficiency will be inreased slightly based on the knowledge of your world size.

However, bounded mode is not always the best choice. Unlike infinite mode, it allocates a certain amount of memory for the area defined (bigger boundaries means more memory will be used). So if your game world is really really huge, you may be better off using infinite mode (since infinite mode only allocates memory for what is displayed on the screen at any given time).

If you tried displaying something through PagedGeometry at this point, nothing would appear. Before anything can actually be displayed on the screen, you need to configure how PagedGeometry is going to represent your trees (or whatever) on the screen. This is done with the addDetailLevel() function:

```
trees->addDetailLevel<BatchPage>(150, 30);
trees->addDetailLevel<ImpostorPage>(400, 50);
```

The first line means all trees up to 150 units from the camera will be displayed with BatchPage (basically no reduction in visual quality, but much faster than using plain entities). The second parameter (30) means that this LOD will transition (fade) out 30 units beyond the 150 unit point. In this case it will be “morphing” into the next LOD. Note that the transition parameter is optional, and should be avoided where possible since transitions will reduce performance.

The second line adds a second detail level, ImpostorPage, which is displayed up to 400 units from the camera. ImpostorPage uses flat billboard images in place of your trees, which are automatically textured to look just like the “real” mesh. Impostors are very fast, but should be used at a distance to avoid the user noticing they are really flat images. Notice that the transition parameter is used here also (50). Since there is no additional LODs after this one, the transition means the impostors will smoothly fade out into the distance for 50 units.

Note that addDetailLevel() is the only function in PagedGeometry that must be called in any specific order; when adding detail levels, you must add them in order from closest to farthest. For example, you can't add a detail level that has a 400 unit range before a detail level that has a 150 unit range.

To review, here's the code so far:

```
PagedGeometry *trees = new PagedGeometry();
trees->setCamera(camera);
```

```
trees->setPageSize(50);
trees->setInfinite();
trees->addDetailLevel<BatchPage>(150, 30);
trees->addDetailLevel<ImpostorPage>(400, 50);
```

Adding Trees

At this point PagedGeometry is ready to display your entities. While PagedGeometry is actually an advanced scene management engine, it's not directly linked with Ogre's SceneManager structure. This means that adding entities to your scene using Ogre's standard functions will not make use of PagedGeometry's optimizations. You have to give PagedGeometry a list of what you want displayed, and it will do so.

You may notice that the PagedGeometry class itself contains no functions for the addition / removal of entities. Instead, you must create an instance of a PageLoader-derived class, and supply that instance to PagedGeometry. You can think of a PageLoader as a list of entities that you give to PagedGeometry (although the internal operation is much more complex and optimized).

The PagedGeometry engine comes with two PageLoader implementations: TreeLoader2D and TreeLoader3D. This tutorial will be using TreeLoader3D (since it's slightly simpler). Depending on which PageLoader you're using the interface will be completely different, but in the case of TreeLoader3D it's as simple as calling treeLoader->addTree():

```
//Create a new TreeLoader3D object first
TreeLoader3D *treeLoader = new TreeLoader3D(trees, TBounds(0, 0, 1500, 1500));

//And add a entity at the desired position/rotation/scale
treeLoader->addTree(myEntity, position, yaw, scale);
```

The first line creates a new TreeLoader3D instance. Notice that the TreeLoader3D constructor requires a pointer to your PagedGeometry object and a boundary. Unlike the PagedGeometry engine itself, TreeLoader3D always requires a boundary in which all your entities will be placed, due to the way they are stored in memory (using a compression technique, so you can store millions of trees in only a few MBs of RAM).

Once the TreeLoader3D is created, all you have to do is add entities to it. As you can see, the addTree() function accepts a pointer to an entity, a position, a yaw value, and a scale value. Note that the entity you supply to addTree() can be added over and over again, since PagedGeometry internally makes a copy of it where necessary. Do NOT make copies of it yourself, since this may be very inefficient both during loading time and during runtime.

After all your entities have been added to treeLoader, you need to assign it to your PagedGeometry object using PagedGeometry::setPageLoader(), like this:

```
trees->setPageLoader(treeLoader);
```

Updating PagedGeometry

Now PagedGeometry is fully configured and ready to render your trees, but there's just one more thing; each frame, you will need to call PagedGeometry::update() for your PagedGeometry instance. This function is used to perform all the caching, batching, impostoring, transitioning, etc that you have configured. If you don't call it every frame, your trees (or whatever) won't appear correctly!

```
[each frame]
{
      trees->update();
}
```

Conclusion

At this point you should be able to start adding trees and other objects to your game, taking full advantage of the optimization features offered by PagedGeometry. When you're ready to learn more, you can read one of the later tutorials, which cover the details of more practical PagedGeometry uses, such as how to trees, bushes, grass, etc to your game.

In the examples folder, you can get a full working example of what you learned in this tutorial by opening the "Example 1 – TreeLoader3D" project