IBR Billboard Cloud Tree Generator

Documentation

Contents

List of Figures

Chapter 1

Reference Manual

1.1 Introduction

The *IBR Billboard Cloud Tree Generator* provides a set of tools creating and preview tree 3d models for being used in real-time applications. The generated tree is thus represented by a set of billboards, called billboard cloud. The billboards are built automatically by a clustering algorithm.

Unlike classical billboards, the billboards of a billboard cloud are not rotated when the camera moves, thus the expected occlusion and parallax effects are provided. On the other hand, this approach allows the replacement of a large number of leaves by a single semi-transparent quadrilateral, which considerably improves the rendering performance. A billboard cloud well represents the tree from any direction and provides accurate depth values, thus the method is also good for shadow.

The billboard cloud decomposes the original object into subsets of patches and replaces each subset by a billboard (See figure 1.2). These billboards are fixed and the final image is the composition of their images (See figure 1.1).

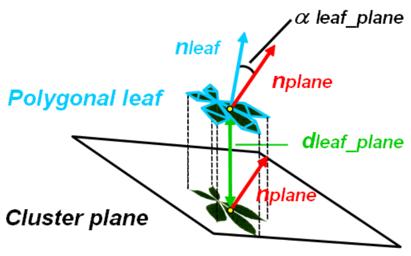


Figure 1.1: Billboard image generation.

The method [1] falls into the class of billboard clouds, but it prepares the billboards automatically and so carefully that the results are satisfactory not only on vehicle scale, but also on human scale, and with some compromises on insect scale as well. Since the billboard cloud model has been proposed to model conventional man-made 3D objects, such as teapots, helicopters, etc., we have to alter their construction to take into account the special geometry of natural phenomena.

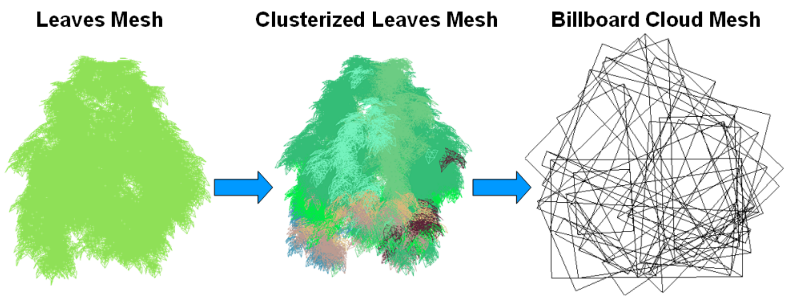


Figure 1.2: Decomposition of the original leaves model.

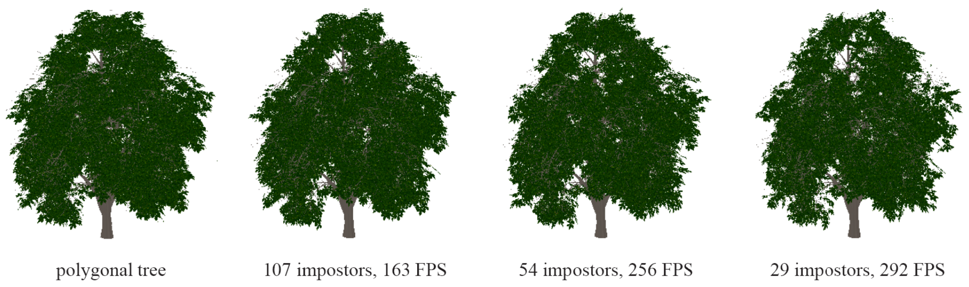


Figure 1.3: Visual and speed comparisons of the original polygonal tree and the tree rendered with different number of billboards.

1.2 Application components overview

The *IBR Billboard Cloud Tree Generator* has been developed using the Ogre3D graphic engine, being compatible with both, the OpenGL and Direct3D render systems, the other optional dependency is OpenEXR. The system has been tested under Windows but there shouldn't be big problems for making it to work under Linux or Mac platforms.

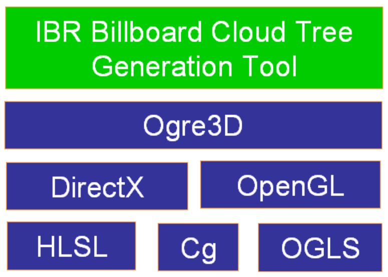


Figure 1.4: *IBR Billboard Cloud Tree Generator* standard dependencies.

1.3 Typical Use Case

Create a tree suitable for real-time rendering must take in care some restrictions, basically the polygon and texture memory used. The sample tree shown 1.5 has 11291 leaves defined by defined by 112910 faces and 338731 vertices. The trunk of the tree has 46174 faces. Current graphic hardware can't handle a forest using this tree model. In these cases this tool is being necessary.

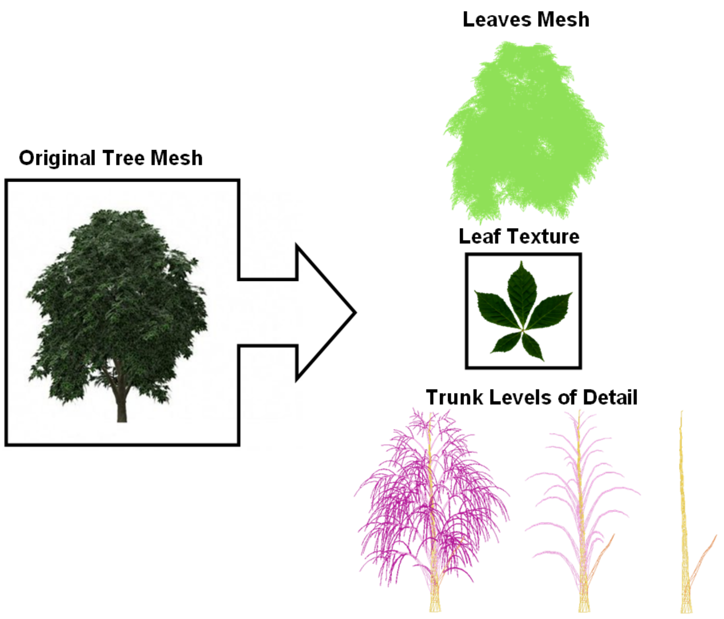


Figure 1.5: Sample polygonal tree decomposed in two models. The leaves model will be processed with the IBR Billboard Cloud Tree Generator.

1. The 3d content creator at first have to create a highdetailed tree with some of the available tools, such as Xfrog or PovTree. (See [3] and [4]). However you can use some of the great Xfrog public plants [5] instead.
2. Then the highdetailed tree model should be decomposed in two, the leaves and the trunk models. (See figure 1.5)
3. The leaves model must be converted into the Ogre3D Mesh file format, there are exporters available for the main 3D DCC, such as Maya or 3ds Max [1].

The trunk model won't be processed with the generator tool and the designer must create a set of levels of detail of the original mesh using a simplification sofware.

The *IBR Billboard Cloud Tree Generator* receives as input the leaves model and the texture used to map each leaf (See figure 1.6).

The output generated is a billboard cloud mesh and three texture atlas as described here:

- The billboard cloud model that replaces the original leaves model. The billboard cloud model will be used in with two techniques described in section 1.6 in the ingame visualization.

[1]Check the Ogre 3D webpage http://www.ogre3d.org in the *Downloads / Tools* section

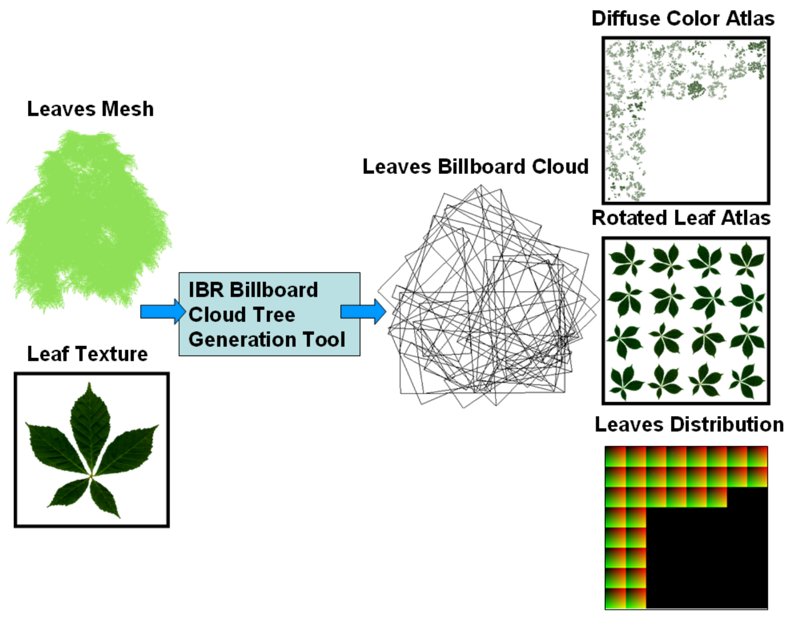


Figure 1.6:

- The texture atlas that contains the diffuse color texture atlas that contains all the textures that will be used with the billboard cloud mesh. This texture will be used by the lowerend technique with standard texture mapping.
- The rotated leaf texture atlas, that contains the leaf placed with different orientations.
- The leaves distribution atlas texture. This texture will be used with the rotated leaf texture atlas in the technique *indirect texturing technique* detailed in section 1.6.1.

The *IBR Billboard Cloud Tree Generator*, is a commandline application managed through configuration files. For each kind of tree we want to process we should create a configuration file.
The application must have as input parameter the configuration file as shown here:

```
IBRBillboardCloudTreeGeneratorCmd.exe -cfg
../../media/chestnut/leaves/sample.cfg
```

These configuration files contain all the parameters you have to specify in order to obtain the desired output.
Those parameters can be sumarized as:

- The media folders parameters.
- The input files parameters.
- The output files parameters.
- The application parameters.

1.4 Media folders

- **Input folders:**

```
# Folder where the sample input leaves model can be found...
Entities Folder ../../media/chestnut/leaves/

# Folders where the sample temporary data will be stored... Entity
Distribution Folder ../../media/chestnut/leaves/

Entity Clusters Folder ../../media/chestnut/leaves/

# Folder where the sample output files will be stored... Billboard
Cloud Folder ../../media/chestnut/leaves/
```

- **Temporary folders:**

```
# Folders where the sample temporary data will be stored...
Entity
Distribution Folder ../../media/chestnut/leaves/

Entity Clusters Folder ../../media/chestnut/leaves/
```

- **Output folders:**

```
# Folder where the sample output files will be stored...
Billboard
Cloud Folder ../../media/chestnut/leaves/
```

1.4.1 Input files parameters

```
# This is the sample input leaves model filename
Entities Mesh Name
chestnutLeaves.mesh

# This is the sample input leaf texture filename
Entity Clusters
Grouped Texture Unit 0 Name chestnutLeaf.png
```

1.4.2 Output files parameters

- **Rotated leaf texture atlas parameters:**

```
# This is the sample input leaf texture filename
Diffuse Color
Entity Texture Name chestnutLeaf.png

# The output texture atlas file name
Diffuse Color Entity Texture
Atlas Name chestnutRotatedLeafAtlas.png

# The output texture atlas bit range (8 bits, 16 bits, 32 bits per
channel)
Diffuse Color Entity Texture Atlas Bit Range 8

# The output texture altas size in pixels
Diffuse Color Entity
Texture Atlas Size 512

# The number of textures with different leaf orientations we want
Diffuse Color Entity Texture Atlas NumSamples 16
```

- **Diffuse color texture atlas parameters:**

```
# The output texture atlas file name
Diffuse Color Texture Atlas
Name diffuseColorAtlas.png

# The output texture atlas bit range (8 bits, 16 bits, 32 bits per
channel)
Diffuse Color Texture Atlas Bit Range 8

# The output texture altas size in pixels
Diffuse Color Texture Atlas
Size 1024

# The size of each billboard texture in the texture
atlas
Diffuse Color Texture Size 16
```

- **Leaves distribution texture atlas parameters:**

```
# The output texture atlas file name
Indirect Texture Atlas Name
indirectTextureAtlas.png

# The output texture atlas bit range (8 bits, 16 bits, 32 bits per
channel)
Indirect Texture Atlas Bit Range 8

# The output texture altas size in pixels
Indirect Texture Atlas
Size 1024

# The size for each billboard texture in the texture atlas
Indirect
Texture Size 16
```

1.4.3 Application parameters

```
# This is the maximum number of billboards desired
Entity Clusters
MaxClusters 1024
```

1.5 Keyboard Controls

Navigation controls:

- **Key F1:** Enable *Diffuse Color Texture Atlas View Mode.*
- **Key F2:** Enable *Indirect Texture Atlas View Mode.*
- **Key F3:** Enable *Rotated Leaf Texture Atlas View Mode.*
- **Key F4:** Enable *Billboard Cloud Final View Mode.*
- **SPACE:** This key iterate along all the billboards generated to see how each group of leaves have been placed on it. This key is working only in the *Diffuse Color Texture Atlas View Mode* and *Indirect Texture Atlas View Mode.*

Navigation controls:
This controls are enabled only in the *Billboard Cloud Final View Mode.* With them you can go around the billboard cloud, using the mouse as the looking forward direction.

- **Key E/Key D:** Accelerate/Brake
- **Key S/Key F:** Turn
- **Key PGUP/Key PGDOWN:** Shift

Special controls controls:
This controls are enabled only in the *Billboard Cloud Final View Mode*. With them you can tweak the indirect texturing leaf texture position. This is explained in detail in section 1.6.

- **Key UP/Key DOWN:** Move up/down the leaf texture.
- **Key RIGHT/Key LEFT:** Move rigth/left the leaf texture.

1.6 Using IBR Billboard Cloud Trees in Games

For using the billboard cloud trees in games, two techniques are shown:

- **Standard texturing technique**: Use only diffuse color texture atlas generated. This technique is for low-end graphic cards.
- **Indirect texturing technique**: This technique uses the rotated leaf texture atlas and the leaves distribution texture atlas in order to have hight resolution rendered leaves.

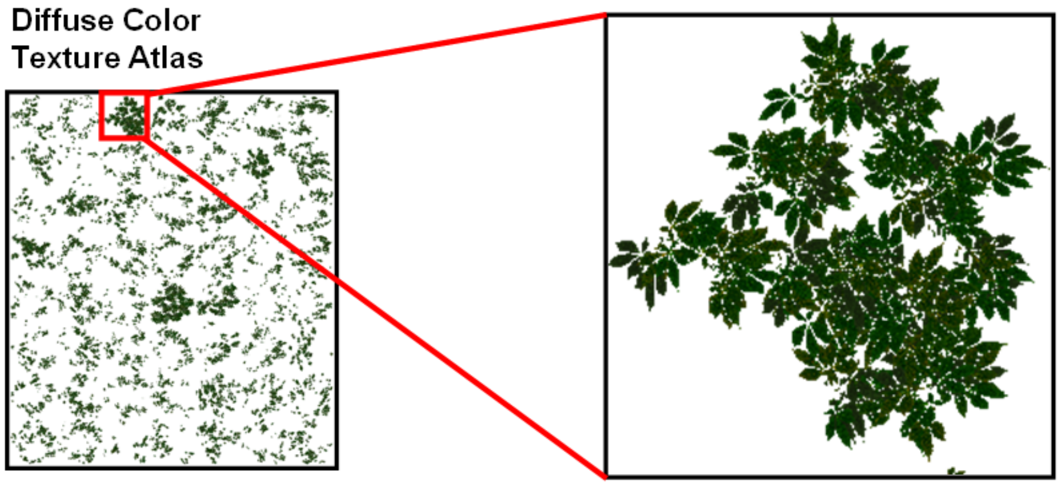


Figure 1.7: Diffuse color texture atlas that contains all the billboard textures. This texture atlas is used in the *Standard texturing technique*.

We will cover only the *Indirect texturing technique* more in detail, because the *direct texturing technique* is straightforward.

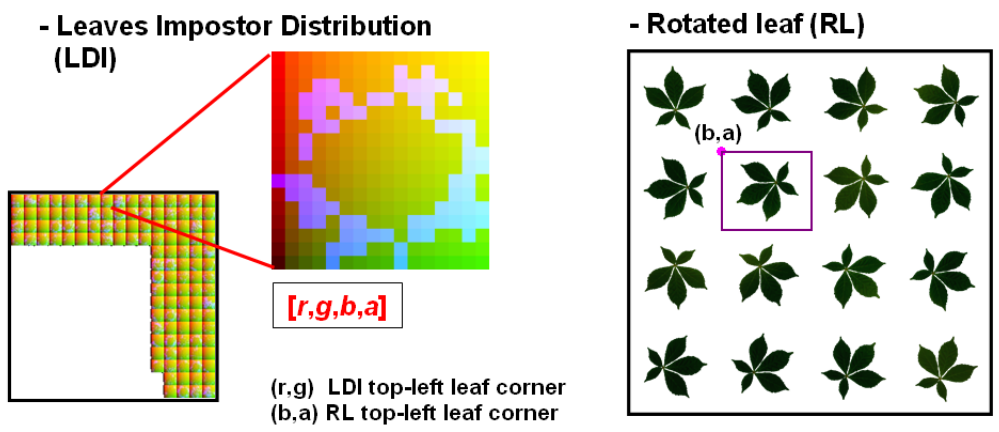

- Leaves Impostor Distribution (LDI)
[r,g,b,a]
(r,g) LDI top-left leaf corner
(b,a) RL top-left leaf corner
- Rotated leaf (RL)
(b,a)

Figure 1.8: Leaves distribution texture atlas and rotated leaf texture atlas. These texture atlases are used in the *Indirect texturing technique*

1.6.1 Indirect texturing technique

The indirect texturing technique is an only one pass technique.

(A) the billboard cloud is the input geometry. The billboard cloud model generated with the *IBR Billboard Cloud Tree Generator* encodes some information in the the *TEXCOORD0* and *COLOR0* parameters.

- **COLOR0:** Encodes the billboard coordinates in the range [0,0] - [1,1]. They are used to compute the current rotated leaf coordinates respect the leaf top-left corner stored in *(r,g)*.
- **TEXCOORD0:** Encode the texture coordinates that tells us where the billboard texture is in the leaves distribution texture atlas.

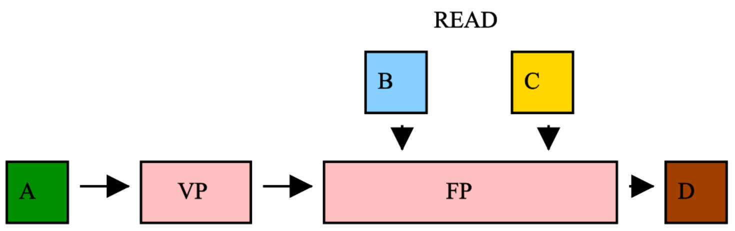


Figure 1.9: This is the diagram showing the indirect texturing rendering pass. Where *A* is the billboard cloud geometry, *B* is the rotated leaf texture atlas and *C* is the leaves distribution texture.

The render state block has the following configuration:

- ALPHAFUNC = CMP_GREATER_EQUAL
- ALPHAREF = 110

- ALPHATESTENABLE = TRUE
- CULLMODE = CULL_NONE

For each texture unit:

(B) Rotated leaf texture:

  - TEXTURE FORMAT = A8R8G8B8
  - MAGFILTER = ANISOTROPIC
  - MINFILTER = ANISOTROPIC
  - MIPFILTER = ANISOTROPIC

(C) Leaves distribution texture:

  - TEXTURE FORMAT = A8R8G8B8
  - MAGFILTER = POINT
  - MINFILTER = POINT
  - MIPFILTER = NONE

1. During the rendering pass, the leaf distribution texture is accessed using the *TEXCOORD0 (u,v)* coordinates.
2. (a) If the value stored in the a channel equal to 0, then this texel is not indexing a leaf position inside the billboard. Therefore will be rendered as a transparent texel above the billboard plane.

   (b) 1. Else if the a channel is not equal to 0, the stored coordinates at *(r,g)* will be the top-left coordinates of the position where the leaf will be shaded.

   2. Using the leaf orientation top-left coordinates, stored in *(b,a)*, we will lookup from the rotated leaf texture atlas the desired one.

```
// Indirect texturing vertex shader
struct VS_OUTPUT {
   float4 Pos:          POSITION;
   float2 texCoord:     TEXCOORD0;
   float2 subTexCoord:  TEXCOORD1;
};


float4x4 worldviewproj;

VS_OUTPUT IndirectTexturing_VS(
      float4 position: POSITION,
      float2 subTexCoord: TEXCOORD0,
      float4 color: COLOR
      )
{
    VS_OUTPUT Out;

    Out.Pos = mul(worldviewproj, position);
    Out.subTexCoord = subTexCoord;
    Out.texCoord = color.xy;
    return Out;
}
```

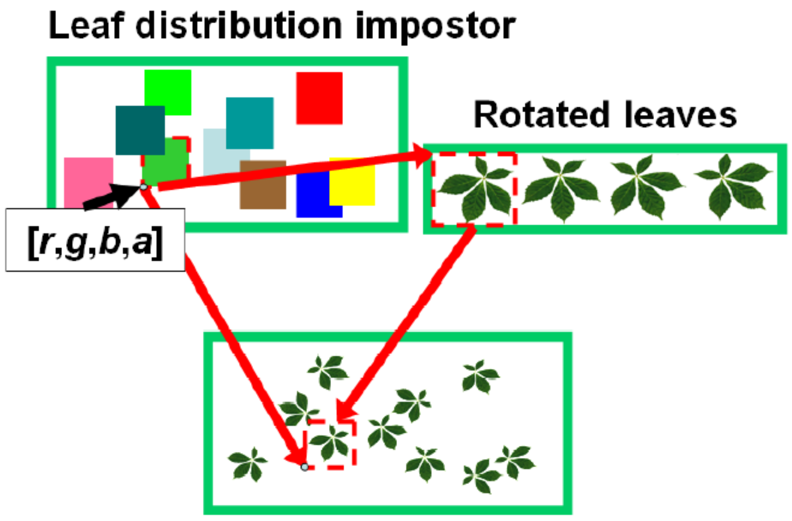


Figure 1.10: Indirect texturing. The billboard is replaced by a leaf distribution impostor and the rotated images of the leaves. All the texels that represents one leaf in the leaf distribution impostor has the same information the (r,g) top-left leaf coordinates and the (b,a) top-left rotated leaf coordinates.

```
// Indirect texturing pixel shader
sampler indirectTexture: register(s0); // leaves impostor distribution texture atlas
sampler sourceTexture: register(s1);   // rotated leaf texture atlas

// epsilon value for the leaf position correction
uniform float epsilonX;
uniform float epsilonY;

uniform float sourceTextureSize;       // billboard texture size in the leaves impostor
                                       // distribution texture atlas

uniform float sqrtNumSamples;          // Sqrt(numberLeaves) in the rotated leaf texture atlas

float4 IndirectTexturing_PS(float2 texCoord: TEXCOORD0,
                            float2 subTexCoord: TEXCOORD1) : COLOR
{
   // Default transparent fragment (fragment without leaf)
   float4 ocolor = float4(0.0, 0.0, 0.0, 0.0);
   // Leaf position correction value
   float2 epsilon = float2(epsilonX, epsilonY);
   // Look-up the leafPositionUVTopLeftCorner and leafRotationUVTopLeftCorner
   // using subTexCoord, which tells where the billboard texture is in
   // the leaves distribution texture atlas
   // * leafPositionUVTopLeftCorner <=> leafInfo.xy
   // * leafRotationUVTopLeftCorner <=> leafInfo.zw
   float4 leafInfo = tex2D(indirectTexture, subTexCoord).xyzw;
   // Leaf UV current coordinates (range [0,0] - [1,1])
   // Using texCoord (billboard coordinates in the range [0,0] - [1,1])
   // we can compute the current leaf coordinates respect the leafPositionUVTopLeftCorner
   float2 leafPositionUVCoord = float2(0.0, 1.0) -
                                abs(texCoord - (leafInfo.xy - epsilon ) );
   // We found a leaf?
   if (leafInfo.w != 0.0)
   {
       // Conversion from [0,0] - [1,1] to [i,j] - [i+(1/sqrtNumSamples),j+(1/sqrtNumSamples)]
       // In order to look-up the leaf with the choosen orientation from the rotated leaf
       // texture atlas
    float2 leafRotationUVCoord = (float2(1.0, 1.0) - leafInfo.zw) +
                         ((leafPositionUVCoord * sourceTextureSize) / sqrtNumSamples);
  ocolor = float4(tex2D(sourceTexture, newcoord).xyzw);
 }
 return  ocolor;
}
```

The billboard cloud trees has been used in two applications:

- Ogre Demo: Developers how will work with Ogre3D should take a special look into this demo. The demo has been tested under OpenGL and DirectX using GLSL/Cg Cg/HLSL shaders respectively.
- RenderMonkey Effect: This demo has the same effects as the the Ogre3D Demo, using GLSL/HLSL shaders. This application should be easiest to understand for those developers who want to use this technique with other game engines and applications.

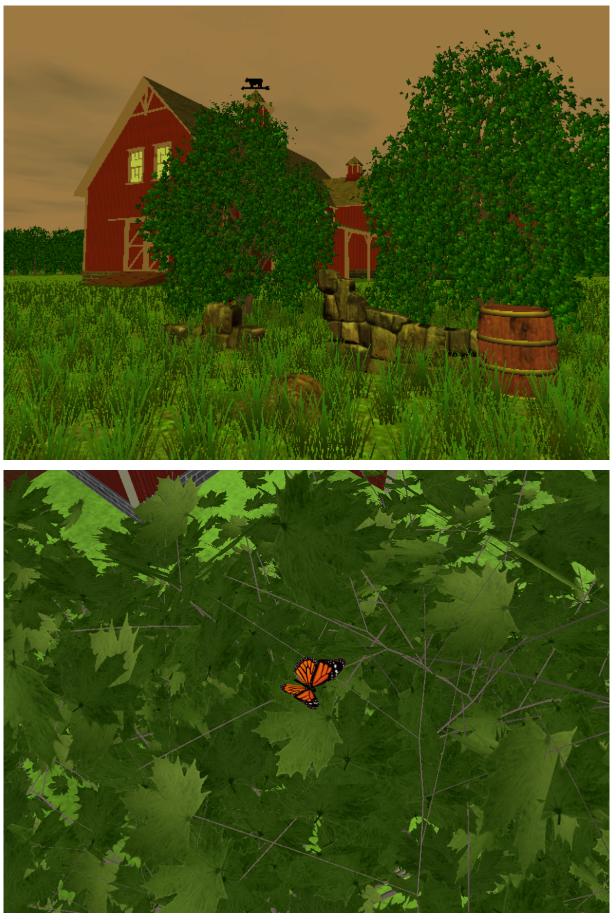
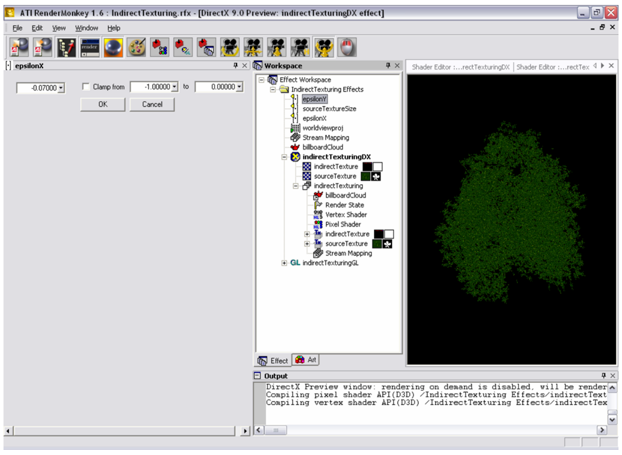


Figure 1.11: Screenshots of the Ogre Demo application.

The indirect texturing technique is good also for shadows. The Ogre Demo has been implemented using Texture Modulative / Additive Shadows and Depth Shadow Mapping.



```
DirectX Preview window: rendering on demand is disabled, will be render
Compiling pixel shader API(D3D) /IndirectTexturing Effects/indirectText
Compiling vertex shader API(D3D) /IndirectTexturing Effects/indirectTex
```

Figure 1.12: Screenshot of the RenderMonkey indirect texturing effect.

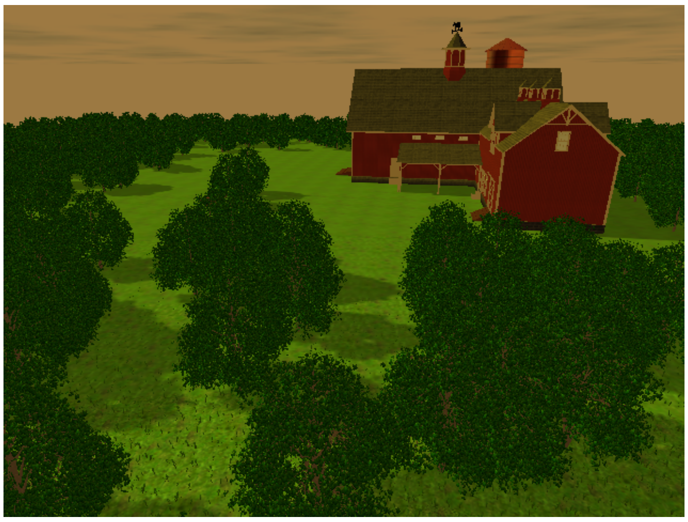


Figure 1.13: Ogre Demo application with texture modulative shadow mapping.

1.6.2 Implementation Issues:

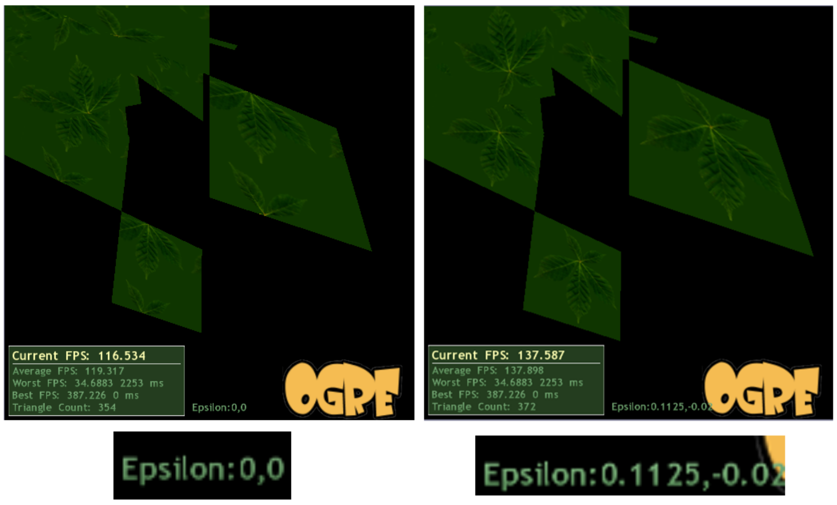
Billboard cloud mesh TEXCOORD0 and COLOR0 parameters:

The billboard cloud mesh generated by the IBR Billboard Cloud Tree Generator can be used for both techniques, the standard texture mapping and the indirect texturing one.

The pixel shader used in the standard texture mapping technique will only use the TEXCOORD0 parameter. And the indirect texturing pixel shader will use both the COLOR0 and TEXCOORD0 parameters.

Incorrect leaf placement with indirect texturing:

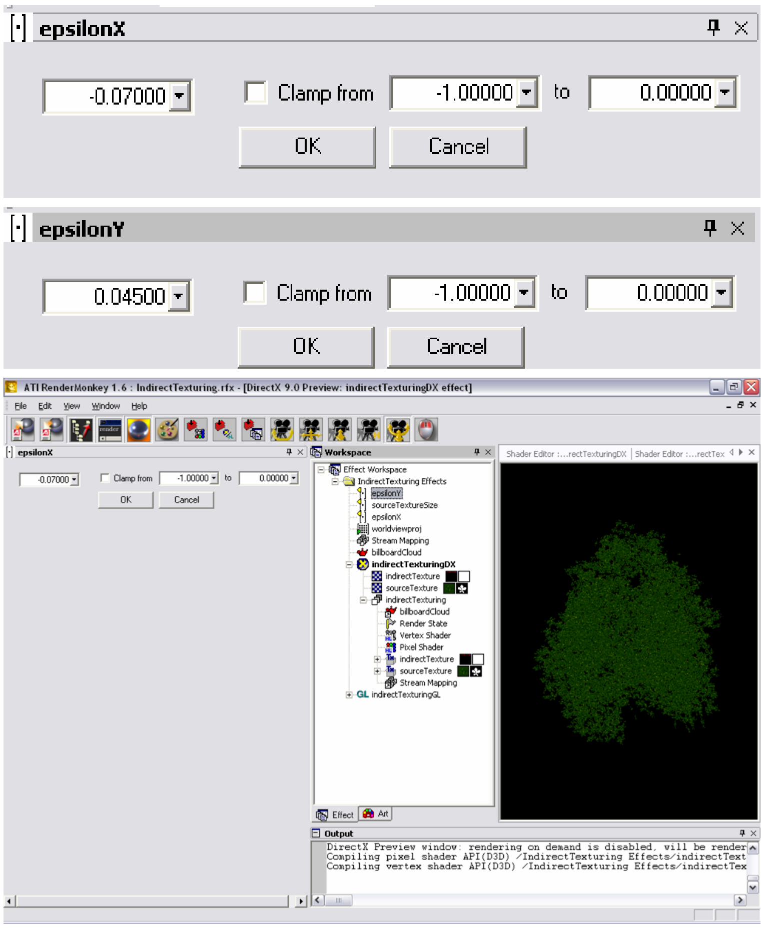
One problem that can appear is due to the wrong placement of the leaf texture, this can be avoided adjusting an epsilon shift displacement.



```
Current FPS: 116.534
Average FPS: 119.317
Worst FPS: 34.6883 2253 ms
Best FPS: 387.226 0 ms
Triangle Count: 354
Epsilon:0,0
```

```
Current FPS: 137.587
Average FPS: 137.898
Worst FPS: 34.6883 2253 ms
Best FPS: 387.226 0 ms
Triangle Count: 372
Epsilon:0.1125,-0.02
```

Figure 1.14: The first picture shows the result of the indirect texturing before fixing the shift displacement that should be used. In the second picture all the leaves are placed correctly. Those views are from the IBR Billboard Cloud Tree Generator, and with the key controls UP/DOWN and RIGHT/LEFT in the Billboard Cloud Final View Mode this small epsilon is applied. The numeric value applied appears at the bottom of the screen. This epsilon should be used as a uniform float2 parameter of the indirect texturing pixel shader. In the RenderMonkey Demo you can tweak this value directly.



```
DirectX Preview window: rendering on demand is disabled, will be render
Compiling pixel shader API(D3D) /IndirectTexturing Effects/indirectText
Compiling vertex shader API(D3D) /IndirectTexturing Effects/indirectTex
```

Figure 1.15: These pictures show how to tweak the epsilon value inside RenderMonkey.

Bibliography

[1] Leaf cluster impostors for tree rendering with parallax. Garca, Ismael, Sbert, Mateu, Szirmay-Kalos, Laszlo. Short Paper of Eurographics (Dublin, Ireland), pp. 69-72, 2005. 7

[2] Tree rendering with billboard clouds. Third Hungarian Garca, Ismael, Sbert, Mateu, Szirmay-Kalos, Laszlo. Conference on Computer Graphics and Geometry, Budapest, 2005.

[3] Xfrog, Greenworks Organic-Software http://www.xfrogdownloads.com/greenwebNew/products/productStart.htm 9

[4] Pov-Tree http://propro.ru/go/Wshop/povtree/povtree.html 9

[5] Xfrog Public Plants http://web.inf.tu-dresden.de/ST2/cg/downloads/publicplants/ 9