ABSTRACT

BORSE, JITENDRA ARUN. Real-time Image Based Rendering for Stereo Views of Vegetation. (Under the direction of Dr. David McAllister)

Rendering of detailed vegetation for real-time applications has always been difficult because of the high polygon count in 3D models. Generating correctly warped images for nonplanar projection surfaces often requires even higher degrees of tessellation. Generating left and right eye views for stereo would further reduce the frame rate since information for a one eye view cannot be used to redraw the vegetation for the other eye view. We describe an image based rendering approach that is a modification of an algorithm for monoscopic rendering of vegetation proposed by Aleks Jauklin. The Jauklin algorithm pre-renders vegetation models from six viewpoints; rendering from an arbitrary viewpoint is achieved by compositing the nearest two slicings. Slices are alpha blended as the user changes viewing positions. The blending produces visual artifacts that are not distracting in a monoscopic environment but are very distracting in a stereo environment. We have modified the algorithm so it displays all pre-rendered images simultaneously, and slicings are partitioned and rendered in a back-to-front order. This approach improves the quality of the stereo, maintains the basic appearance of the vegetation, and reduces visual artifacts but it increases rendering time slightly and produces a rendering that is not totally faithful to the original vegetation model.

Real-Time Image Based Rendering for Stereo Views of Vegetation

by

JITENDRA BORSE

A THESIS SUBMITTED IN PARTIAL
FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE
COMPUTER SCIENCE
AT
NORTH CAROLINA STATE UNIVERSITY
RALEIGH, NC
2002

APPROVED BY:

_____

Dr. David McAllister
Chair of Advisory Committee

_____ _____

Dr. Christopher Healey
Member of Advisory Committee

Dr. Robert Rodman
Member of Advisory Committee

DEDICATION

To my *Parents*

To my *Uncle*

Who have made my higher education dreams come true.

BIOGRAPHY

Jitendra Borse was born in Dhule, Maharashtra, India. He did his undergraduate program at Sardar Patel College of Engineering, Mumbai, India and received a B.E. in Computer Engineering from University of Mumbai in 2000. He joined North Carolina State University in the fall of 2000 to pursue MS in Computer Science. He worked under Dr. David McAllister since September 2000 on the topic Real-time Rendering of Vegetation. As a part of his MS co-operative program, he has worked at Duke UNC Brain Imaging and Analysis Center at Duke University, Durham, North Carolina.

ACKNOWLEDGEMENT

I would like to thank Dr. David McAllister, my advisor, for his guidance, and motivation and for providing the opportunity to work with him. Dr. McAllister has been instrumental in bringing me to the field of computer graphics, and stereo and I am glad I got the chance to work with him.

I would like to thank Dr. Robert Rodman for being on my thesis committee and providing invaluable advice and support in so many ways over past two years. I would like to thank Dr. Christopher Healey for accepting to be a committee member and providing useful comments on this thesis from time to time.

I would like to thank my friends for being there when I needed them. Special thanks to Nihar, for his help in documenting this work.

Finally, I would like to thank my parents and family for their support and love. Thanks Mom for all the encouragement, you are the reason I am here. Thanks Dad for faith in me. Thanks Uncle for everything. Words alone cannot express the thanks I owe to my family, for their moral and emotional support.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Motivation

Traditional computer graphics use 3D models to suggest a scene. Over the years, considerable efforts have been made to generate models that look realistic. These efforts have been successful, but at the expense of generating more complex models most of which have polygons as small as a pixel. Such models are computationally expensive with no upper bound on per frame computation requiring specialized rendering hardware. However, most applications demand real-time frame rates, in particular virtual reality environments.

Indeed, the motivation to study this problem arose from suffering from the vegetation rendering speed in a hemispherical projection environment. To try to attain real-time frame rates, very simple tree models were required for a low altitude flight simulation. Lollipop-like models were used for trees, but the resulting scene realism was not acceptable. In addition, the frame rate in this hemispherical dome environment dropped drastically when even simple 3D models of vegetation were introduced in a scene. The high polygon content and the amount of details of 3D models of vegetation make real time rendering difficult in such an environment. In addition, warping images to correctly display on the dome required high degrees of tessellation.

1.2 Image Based Rendering

Given these complexities, we look at alternatives to using 3D models directly. An obvious solution to this problem is drawing as few polygons as possible. Our objective was to develop a system that would display high quality images at a constant and interactive rate, independent of scene complexity. Image-based rendering (or representation, IBR) has recently gained popularity and is currently a topic of active research. IBR has come up as an efficient way of generating novel views of an object /scene. IBR methods improve performance by rendering 2D objects in a way that makes them appear to be 3D. In fact, sometimes it's possible to replace an entire 3D scene with a single polygon.

The main advantage of using image-based rendering methods over 3D models is that it is computationally less expensive, and rendering time is usually constant and not dependent on object complexity. With traditional rendering techniques, the time required to render an image increases with geometric complexity of the object. IBR replaces the three dimensional scenes with a set of images and rendering the scene is therefore independent of the scene complexity. IBR thus decouples scene complexity from rendering complexity by doing work proportional to the number of pixels in the final image instead of work proportional to the number of polygons in 3D model. Thus, higher frame rates can be attained. It should be noted that a constant and reliable frame rate is highly desirable in a real-time application.

Image-based techniques have a number of attractive properties; however, certain issues still pose a problem. These problems include image resolution, number of images stored per object, and occlusion related artifacts. Using a large number of images to represent a single object would result in memory overhead. However, even with a large number of images there is no guarantee that the complete surface of an object has been captured. Surfaces

that were occluded in the sampled images but visible in some intermediate view produce holes in the final image, which is not acceptable. The algorithm developed needs to eliminate all these drawbacks and specify an efficient method to obtain images.

1.3 Slicing

This work attempts to explore how the slicing technique proposed by Jauklin[1] can be modified to generate stereoscopic images in real-time and how to eliminate or reduce the effect of certain visual artifacts which are not very evident in monocular viewing but become obvious in stereoscopic views. The original (and modified) algorithm takes advantage of the lack of human ability to capture and remember minor details of a complex, moving object like tree. The viewer views the tree as a whole object rather than focusing on minor details like individual leaves. Vegetation also has holes through which objects behind it can be seen. All these complexities enable using simple manipulation and approximation to the original model.

1.4 Objectives

Our goal is to produce acceptable stereo images of forests of vegetation in real time and to produce views that appear "natural" in a walk-through environment. The images reproduced by our modified algorithm maintain the "treeness" property of the vegetation but they are not faithful to the original 3D model. The current implementation requires the viewer to remain at approximately the same height from the ground. There is a heavy constraint such that the viewpoint cannot be at a higher elevation than the highest point in the vegetation. This algorithm can hence be used in driving simulations, which

have similar height constraints, but cannot be used for applications where an aerial view of the vegetation is required.

Stereo images are presented in the appendix for cross viewing[2]. Left and right eye images have been produced using off axis perspective projection. The model used as an example for this paper is shown in Fig.1. The performance statistics are based on a Pentium IV 1.7GHz processor and a NVIDIA Quadro2 MXR/XE graphics card.

Figure 1 Original Tree Model

Chapter 2
Previous Work

2.1 Texture Mapping and Billboarding

Texture mapping is the simplest form of image-based rendering. An image is mapped onto a surface in the scene. Billboarding[3], a technique that uses texture mapping simply maps texture onto a polygon, which orients itself according to changing view direction. The polygon is called a billboard. The billboard rotates around an axis and aligns itself to face the viewer as much as possible. In fact, billboarding of vegetation is a popular technique in representing trees in real-time applications and is very effective when the object is very far from the viewer. It is extensively used in a number of games. Simple texture mapping is further extended to map textures onto surfaces using environment maps or bump maps among others. These techniques improve the appearance of image textures and the surface.

A scene containing more polygons will require more processing time each frame, and as a result will give a lower frame rate. If the object can be represented with fewer polygons, frame rate would improve. The objective is to draw as few polygons as possible. Billboarding is a step in that direction but the poor realism offered by billboarding is not acceptable. However, we explore some variations of conventional billboarding in search of acceptable results. Considerable attention has be paid to generating image based representation of objects in a way such that, the objects can be completely reconstructed using these images. Image based representation has not been restricted to single object; it has been explored to represent a entire scene.

2.2 View Interpolation

Chen and Williams[4] carried the texturing concept further by using interpolated images to portray three-dimensional scenes. Their method exploits the fact that a sequence of images from closely spaced viewpoints would be highly coherent. Each image stores a set of camera parameters and a depth buffer. Using each pixel's screen coordinates (x, y and z) and the camera's relative location, a correspondence between the pixels in each pair of images is established by a 4x4 transformation matrix. The transformations are reduced to 3D spatial offset vectors for each of the pixels. The offset vector indicates the amount each of the pixels moves in its screen space as a result of the camera's movement. The offset vectors are pre-computed and stored in a morph map, which represents forward mapping from one image to another. Using these maps, smooth interpolation between views is used to obtain the image from the current viewing position. These maps are better suited for smooth textured surfaces but not apt for objects like vegetation, which has high depth complexity.

2.3 Pre-computed Z-buffer views

Max and Ohsaki[5] modified this approach and used z-buffer views instead of images used by Chen and Williams. Z-buffer views are more suitable for finely divided objects like trees, where depth coherence at adjacent pixels is limited. The method stores a number of pre-computed z-buffer images from preset viewing directions. Using this depth information a 3D point for each image pixel is generated, and these 3D points are rotated into proper position and projected onto an output z-buffer. The reconstruction may result in pixels missing in the final output image. The undefined pixels are left transparent, so that the background is seen as gaps in the leaves. One significant advantage of this method over most other image-based rendering algorithms is that normal and color information are also encoded so that the shading an be done as a

post process. However, this pixel-by-pixel reconstruction technique is too slow to be acceptable for real-time applications.

2.4 QuickTime VR

Chen[6] presented an image-based system for creating and interacting with a virtual environment. The system uses environment map, in particular panoramic images, to compose a scene. The environment maps are orientation-independent images, which allow the user to look around in arbitrary view directions. Multiple environment maps are linked together to define a scene. Cylindrical images offer advantages because capturing a cylindrical panorama is easier than other types of environment maps. The system digitally warps one of these cylindrical images to obtain a new view. A 360-degree cylindrical image can be created by computer rendering, specialized panoramic cameras, or by stitching together overlapping photographs taken with a regular camera.

Camera panning, tilting and zooming is simulated by using image warping techniques. The user moves in the scene by jumping through the maps. The method defines a two-dimensional or three-dimensional array of environment maps. Viewpoints in space are quantized to the nearest grid point to approximate the motion, as shown in Fig. 2.

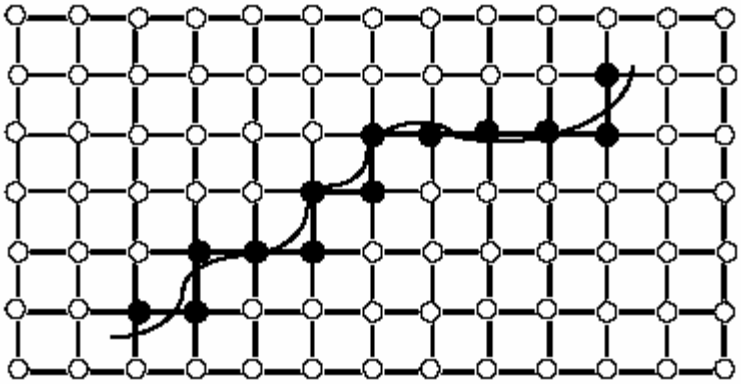


Figure 2 An unconstrained camera path and an approximate path along the grid lines

All the processing is done on-the-fly using a software-based real-time image-processing engine. This image-based approach was implemented in the commercial product, Quicktime VR, built on top of Apple Computer's Quicktime digital multimedia framework. If only the view direction is changing and the viewpoint is stationary, this method offers a very efficient solution. However, moving freely in the scene is difficult, requiring large number of environment maps, and defining smooth interpolation between a set of environment maps. This technique serves the purpose very well if used for backdrops providing immersive quality, but not very useful to represent a single object.

2.5 Layered Depth Images

Another approach, which can be considered an extension to texture mapping, is the use of sprites. A sprite is an image that moves around the screen. Consider the scene as a series of layers. Each object is represented to lie in one of the layers. Each layer has depth associated with it. The sprites are then rendered in back-to-front order, without the need for a z-buffer. As the viewer moves perpendicularly to the direction of view, the layers can be moved relative to their depths. However, this would work well only for points that are very close to the original viewpoint from where sprites were generated. Layered Depth Image[7] seeks to improve the basic sprite algorithm providing a solution to

handle more disocclusions and improve parallax. Each pixel in an image is represented by multiple depth pixels. Instead of storing a 2D array of depth pixels, a 2D array of layered depth pixels is stored. A layered depth pixel stores a set of depth pixels along one line of sight sorted in back to front order. The front element in the layered depth pixel samples the first surface seen along that line of sight.

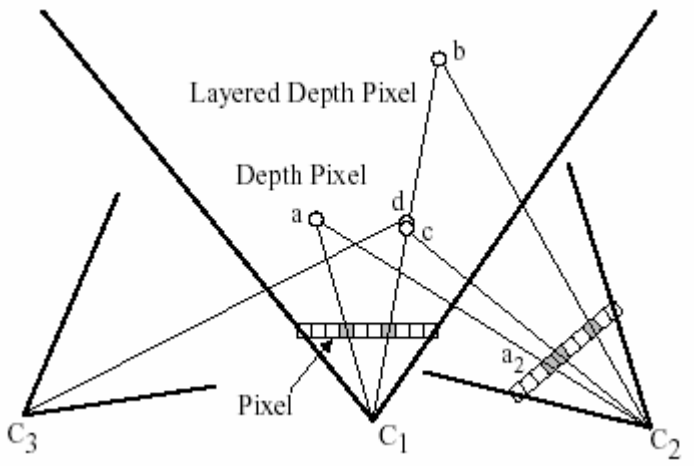


Figure 3 Layered Depth Images

The method uses a incremental warping algorithm to create an output image from a new camera position. It defines a camera matrix composed of an affine transformation, a projection matrix, and a viewport matrix for each camera position. Given a new desired camera position, a transfer matrix, $T_{1,2}=C_2C_1^{-1}$ is generated where C_1 is the LDI camera's matrix and C_2 is the output camera's matrix. Given image co-ordinates of a point (e.g., a in Figure 5) seen in LDI camera the transfer matrix computes image co-ordinates as seen in the output camera (e.g., a_2 in Figure 5). However, the results as stated provide a low frame rate between 4 to 10 frames per second, and the data file generated is too large.

2.6 Layered Impostors

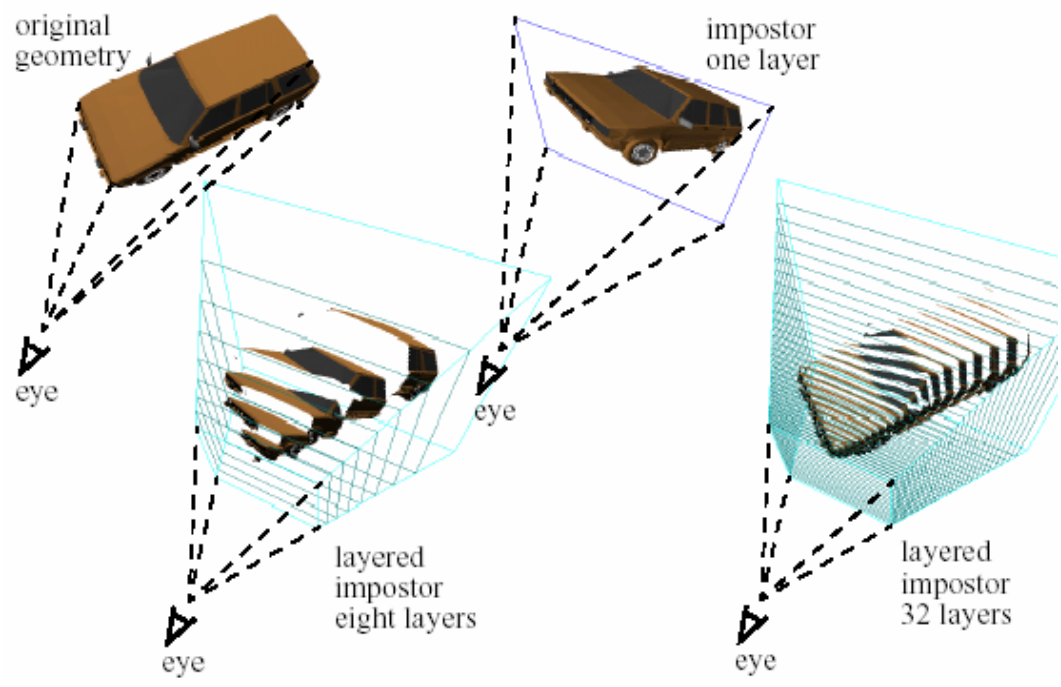


Figure 4 Layered Impostors

Another method uses layered impostors[8], but it focuses more on representation. It is similar to a single slicing, but with a higher number of layers per image. This technique warps the layered impostors for a new viewpoint, with each pixel in the image grouped based on its depth. A single image with multiple layers is used. Increasing the number of layers, with some objects requiring as high as 64 layers, can increase the warping accuracy. For an observer near a viewpoint closer to the sampled viewpoint, a single image might be sufficient. Schaufler explored the multi-layered imposter technique[9] further. He investigated storing impostors from multiple viewpoints and using the one that was generated with a viewing direction closest to the current viewing direction. However to avoid holes, multiple images can be warped to fit the desired view. Schaufler recommends sampling images from about 20 viewpoints evenly distributed over a sphere. However, trees are not good candidates for image caching due to their high geometric complexity.

2.8 Slicing

Aleks Jauklin proposed a method called slicing that uses alpha-blended textured polygons. Pre-rendered images, called slices, are generated from six different viewpoints. Depending upon the angle of the observer, the two closest viewpoints among the six are chosen and appropriately blended. This method is very simple and able to generate frames at an interactive rate. We investigate this method and suggest appropriate modifications to overcome visual artifacts that result when rendering in stereo. This method is explained in details in chapter 4.

Chapter 3

Billboarding

3.1 Introduction

A billboard is a flat surface (panel, fence, wall, etc) on which signs or advertisements are posted. In a 3D graphics context, it would be just a logical extension to the above concept. A billboard in this context is defined as a polygon onto which a texture is mapped and the technique of doing this is known as billboarding. It is one of the most elementary of IBR methods. The texture is normally an image of a complex object, and the polygon is a rectangle whose orientation is changed as the viewer moves around the scene. The billboard rotates such that it faces some target, camera, another object or some point in the scene. Generally, it faces the camera or the viewer. However, certain applications might require the billboard facing a target other than the viewer. For example in games, a football player will always face the ball.

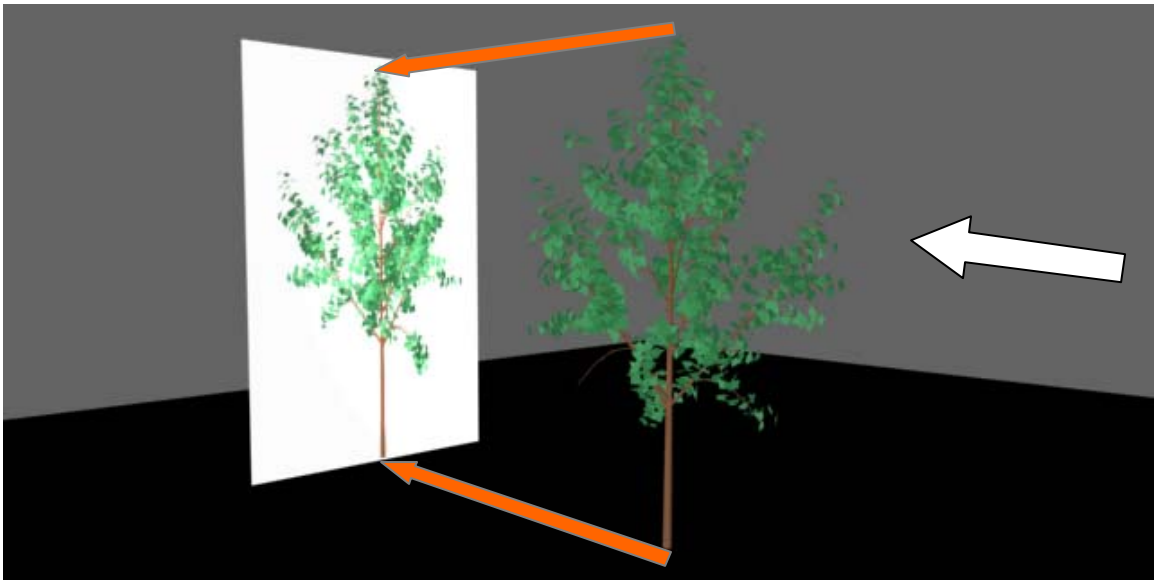


Figure 5 Generation of Billboard

Fig.2 explains this concept of generating billboard and Fig.3 illustrates how to use it in the scene.

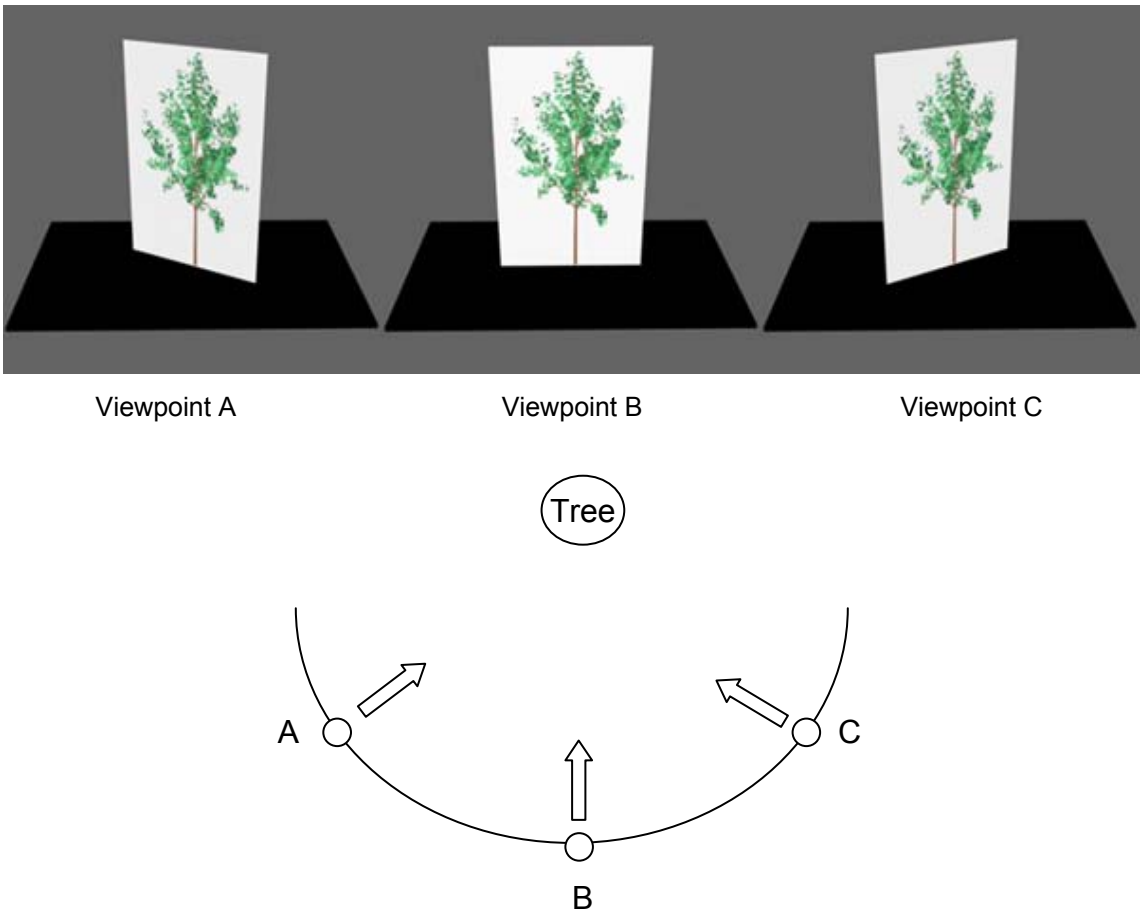


Figure 6 Rotation of the billboard as viewer moves around the object (tree)

Billboarding is useful in applications for rendering objects with a fairly high complexity. This technique is quite popular in games, Virtual Reality (VR), and other applications that require rendering a large number of polygons at a high frame rate. Billboarding can be used to cut back the number of polygons required to render an object by using a texture instead of a 3D model. Billboarding allows us to replace the geometry with a single texture and change its orientation in such a manner that the 2D objects appear to be 3D. Apart from reducing the number of polygons, billboarding can be used to represent many phenomena that do not have solid surfaces like smoke, fire, explosions and

clouds. It can be used to achieve some goal related to the purpose of an application e.g., some application might require a particular object to always face a particular target, where the target can be the viewer or another object in the scene or a particular direction.

Billboarding can be adapted and implemented with several variations. Fig.2 illustrates the most basic billboarding method. A tree is billboarded into the scene. Trees are one of the most common objects billboarded in many applications, especially in games where trees are used more in the background rather than in the foreground. A single tree image is obtained and mapped onto a rectangle. The source of this image can be a 3D model or a photograph of real vegetation. As the viewer moves around in the scene, the billboard is rotated about a vertical axis through the trunk to face the viewer.

Billboarding is best suited for symmetrical objects. Objects like trees are said to be cylindrically symmetric for all practical purposes, with a vertical axis through the trunk of the tree being the axis of symmetry. Such objects allow rotation only about a vertical axis. They are rotated about such an axis to reorient the billboard as the viewer moves. Objects like clouds, bushes, and smoke have spherical symmetry. Such geometries allow billboards to be rotated up and down along with right and left i.e., the billboard can be rotated about any arbitrary axes as it has infinite axes of symmetry.

Depending on the freedom of rotation allowed, billboarding can be classified as cylindrical or spherical. For spherical rotation, there is no restriction to the orientation of the object, whereas in the cylindrical approach, the rotation of the object is restricted to a fixed axis, usually the positive direction of the Y-axis. DirectX SDK suggests using multiple billboards moving independently for effects like smoke, clouds, etc. Each part of a cloud can be a rectangular primitive. Each primitive of such groups can be allowed to move independently giving it a dynamic appearance. Smoke can be simulated in a similar manner.

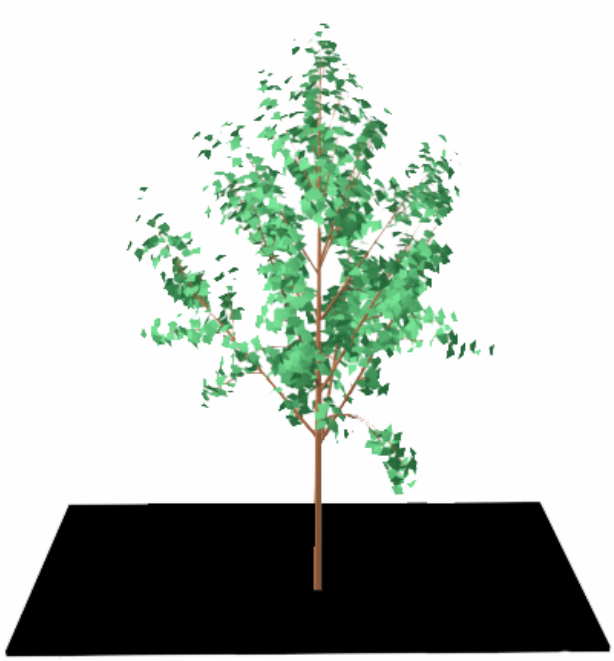


Figure 7 Billboard with transparency

Selective portions of the billboard can be made transparent. In Fig.4, portions of the billboard where vegetation is not present are transparent. The parts of the billboard image that one doesn't want to display can be handled appropriately by making corresponding portions of the billboard transparent. How transparency is handled is an implementation issue. For images obtained by rendering 3D models, the model can be rendered with a white or a particular colored background (take care that this color is not present anywhere else in the vegetation) and drawing pixels can be drawn with this chosen color transparent e.g., if possible, RGBA images can be used with alpha values set to zero for pixels not belonging to vegetation.

3.2 Limitations

While this technique works quite well and enables very fast rendering, it is limited by the fact that the viewer is always presented with the same view of the vegetation. Regardless of his position, he sees the same view and when viewed from a closer distance the planar nature of such geometry becomes evident.

The above discussion suggests that basic billboarding cannot be used directly to render trees to produce continuously changing views of the tree while the viewer is in motion. Certain variations were tried to find out if any modification to the basic algorithm could be used to obtain a realistic effect. In one such approach, the tree was pre-rendered from sixty different viewpoints placed along a circle around the tree. The points were spaced equally angularly, separated by six degrees, one image generated per viewpoint. Sixty images per tree is a large number; however, this was just the initial test to check whether any such variation would work. For every frame, two images closest in angle to the current viewpoint were determined and blended to produce the view of the vegetation. Alpha values of each of the two chosen images depend on its closeness to the viewpoint. However, the blending of billboards produced unacceptable ghosting or shadowing effects. Even with just a six-degree separation between the two blended images, a pop-up effect was very evident. This method was abandoned.

3.3 Billboarded Forests

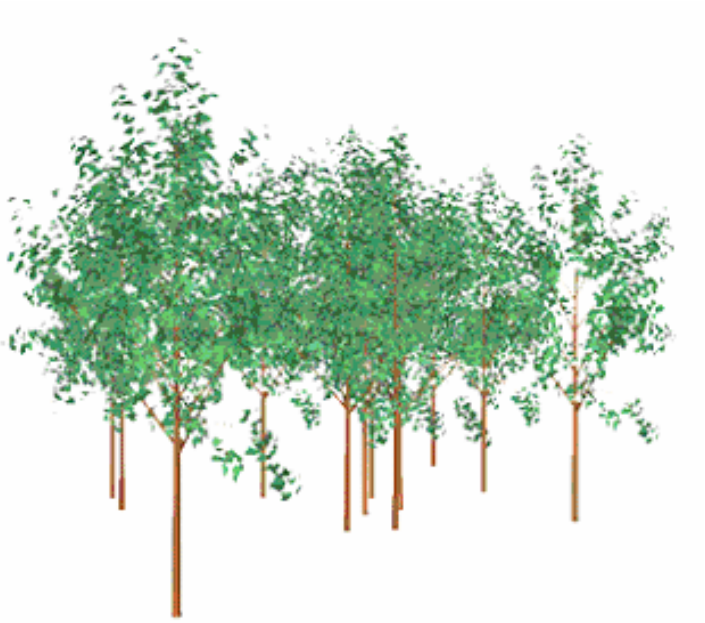


Figure 8 Forest using billboarding

The discussion so far has focused on rendering a single tree in a scene. The use of billboarding seems limited by the fact that the viewer sees the same view of the vegetation. However, certain properties inherent to vegetation, namely density and transparency, and motion parallax due to viewer motion can be used to an advantage while rendering a forest. The high detailed vegetation is made complex because of the fact that we can see certain objects behind the vegetation through the holes in it. When many such trees are introduced in the scene very close to each other, the attention of the viewer will wander, hiding the fact that the same view of the vegetation is used. This experiment was tried using the same 3D model to generate different images. Images were repeated with a different texture size. The basic billboarding still worked very well, and the stereo generated was acceptable. This suggests billboarding does not work to render sparsely spaced trees but works very well to render forests, generating very good stereo. The next chapter introduces an algorithm to

render sparse trees in the scene. It also offers a real-time solution for rendering multiple trees.

Chapter 4
The Jauklin Algorithm

4.1 Slicing

Billboarding enables real-time rendering of complex entities; however, the quality of images offered by billboarding is not acceptable in an environment where high scene realism is desired. Aleks Jauklin introduced an image-based multi-layered representation of vegetation, which provided significant improvement in quality over billboarding, offering high quality realistic images of vegetation. This method, known as slicing, uses a set of pre-rendered images to represent vegetation from a viewpoint. Such images are generated from multiple viewpoints, which are selectively blended to obtain a view of the vegetation from any random viewpoint approximately at the same elevation above the ground.

Slicing offers a real-time solution with acceptable quality, enabling us to render multiple trees in the scene. The method takes advantage of the complexity of vegetation, limitations of the human eye, and motion parallax to generate a close approximation to the original 3D model used to generate the images. This method seems to offer satisfactory image quality and frame rate required for real-time rendering. However, certain visual defects, not easily visible in a monoscopic view but unacceptable in stereo have to be corrected. This chapter presents the original algorithm introduced by Jauklin followed by anomalies, which need correction. Modifications to the algorithm to eliminate or reduce the anomalies are presented in the following chapter.

4.1.1 Sparse and Solid Entities

The Jauklin algorithm combines traditional mesh-based and comparatively new image-based rendering approaches. Vegetation is separated into solid entities and the sparse entities, which structurally form two different units. Trunk and limbs form a solid entity and the crown comprising the sparse entities of leaves and twigs.

Figure 9 Sparse entities and solid entities

The solid entities have a low polygon count, enabling the use of a mesh based rendering scheme. Mesh simplification methods are used to further reduce the complexity depending on the required level of detail. Sparse entities composed of leaves and twigs contain most of the complexity of vegetation. A high level of detail and visibility of the background behind the tree makes it possible to use an image based rendering method even though it offers only an approximation of the vegetation. Sparse entities are represented using a set of parallel images known as slices. Jauklin defines a slice as a planar layer, represented with an ordinary alpha or color-keyed texture. A set of such parallel, equidistant slices is known as slicing. All such slices are generated

from the same viewpoint. When used in a scene, slices are drawn in back to front order without the need of Z-buffer.

4.1.2 Generation of slices

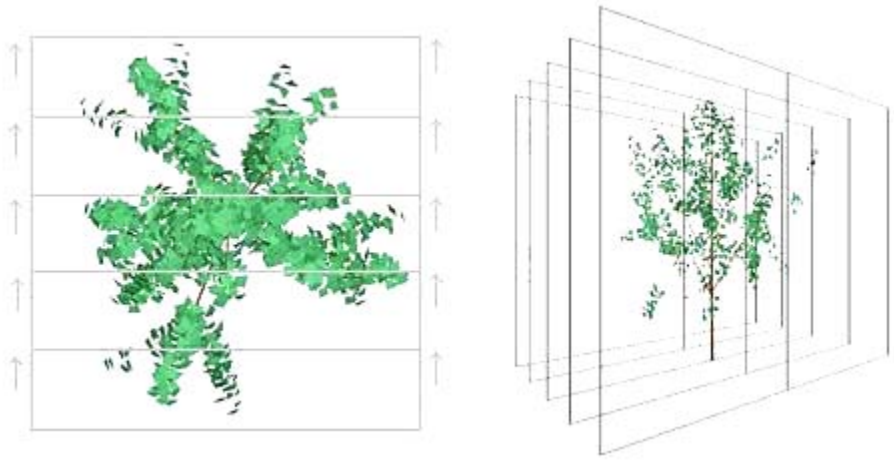


Fig. 7(a) **Fig. 7(b)**

Figure 10 Generation of slices

A bounding box is defined to nclose the entire vegetation. For any generic viewpoint, the bounding box is oriented such that the top and the bottom planes of the box are parallel to the ground and the viewing direction perpendicular to the front face of the bounding box. The viewing direction is defined by a vector from the viewpoint to the trunk of the tree and parallel to ground. Six parallel, equidistant planes with dimensions the same as the front plane (the plane visible from the viewpoint), are placed so that the first and last plane coincide with the front and the back face of the bounding box. Any two adjacent planes are treated as near and far clipping places, and the hidden surfaces in between them, with respect to the viewer, are eliminated. The visible surfaces that remain are projected orthographically on the back-clipping plane. The complexity of vegetation enables us to use only five layers and convey the necessary depth information.

The Jauklin algorithm renders the trunk and the crown separately. The trunk is rendered using mesh-based simplification. This rendering takes place while the application is being executed. The crown is pre-rendered to generate slices, which are used by the application as textures. The color and lighting for this online rendering of trunk and the offline rendering of the crown must be matched to hide the difference in environment. Aleks Jaulkin suggests computing textures by pre-computing the vertex lighting in a radiostiy package for the whole tree. This pre-computed vertex lighting data must then be used to render textures in the application**.**

The tree is pre-rendered from six different viewpoints around the tree at the same height from the ground. All six points are equally spaced angularly at the same horizontal distance from the tree. Each slice in every slicing has the same dimensions, and the distance between adjacent slices is the same for every slicing. The orientation of slicing is fixed and independent of the observer's position.

4.1.3 Blending

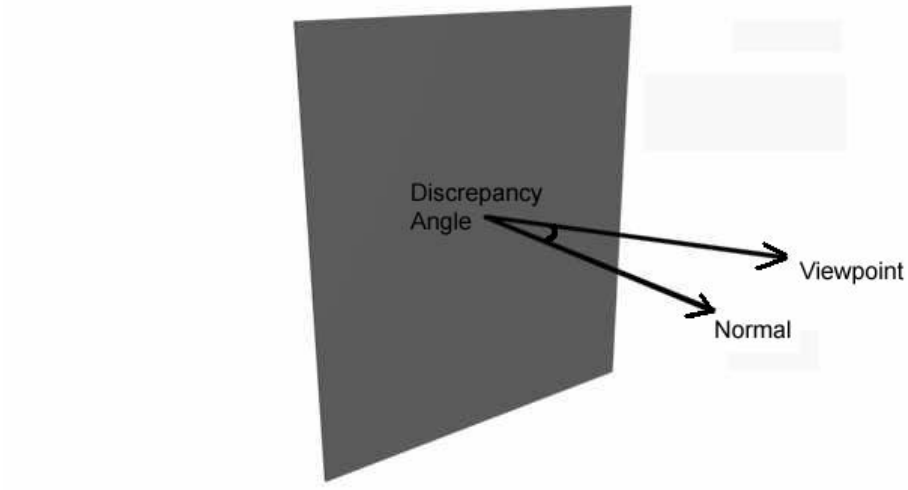
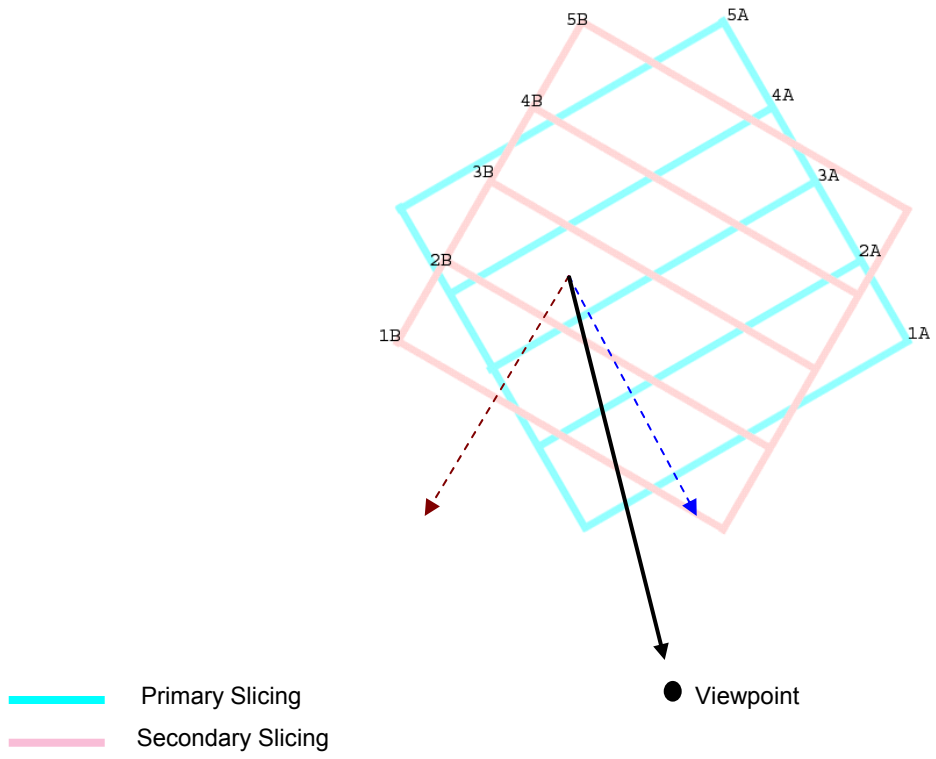


Figure 11 Calculation of discrepancy angle

Two adjacent slicings are used to generate views of the vegetation from any arbitrary point. Note that this algorithm works well only for points that are approximately at the same relative height. Hence, further references to an arbitrary point would mean an arbitrary point at a particular height, unless mentioned. The algorithm determines the slicings closest and second closest to the viewpoint. The author defines the angle between slicing and the direction of viewing as the discrepancy angle. The slicing having the lowest discrepancy angle is known as primary slicing and the second lowest as secondary slicing. The levels of transparency for both the slicings are varied. The transparency of the slicings is proportional to the discrepancy angle. The transparency varies between zero and one.



5B - 5A - 4B - 4A - 3B - 3A - 2B - 2A - 1B - 1A

Figure 12 Order of rendering the slices (top View)

The primary and secondary slicings are blended using the calculated transparencies. Each slicing must be drawn in the correct back-to-front order. First, the farthest secondary slice is drawn, followed by farthest primary, second farthest secondary, second farthest primary, continuing so until all slices belonging to the two slicings are drawn. Fig.8. illustrates the ordering.

This practical method enables interactive rendering of vegetation. The results achieved with respect to number of trees and frame rate is acceptable for a real-time application like VR. The multi-layered images encapsulate the required depth information. This provides the necessary depth perception to give the viewer the feel of a 3D model.

Figure 13 Alpha Blending of slices

4.2 Anomalies

The algorithm works very well for monoscopic viewing; however, it has certain inherent anomalies, which are unacceptable for viewing in stereo. In monoscopic viewing these visual artifacts can be ignored or go unnoticed.

These artifacts can be attributed to the use of blending, however smooth, and to change in the order of slicings being drawn on the screen.

The slicings drawn are partially transparent with their transparency levels varying as viewer moves around the scene. The transparency varies between 0 and 1. As slicing is introduced into the scene, it starts with opacity of 0, gradually increasing to 1 and finally decreasing to 0 and disappearing. When emerging (alpha increasing), parts of the vegetation appear as if moving out of fog. Similarly, when a slicing fades out (alpha decreasing) those corresponding parts of the vegetation begin to disappear. This particular phenomenon is very distracting in stereo. In monocular viewing, due to a lack of depth perception, one does not notice the part of vegetation in front which is disappearing.

Appearance and disappearance are important issues, which need to be corrected before adapting this method in stereo. Besides this, there is a change in order in which the primary and secondary slicings are drawn. As the viewer continues to move around the tree, at a certain point, the discrepancy angle becomes equal for both the primary and the secondary slicing. At this point, the primary and secondary slicings are interchanged. There is a change in the order in which the slicings are drawn. This switch is easily noticeable, in stereo. When swapping occurs, leaf and branches that were in front in previous frames appear to be at the back and vice-versa.

These limitations need to be corrected to implement this algorithm in stereo. The next chapter suggests some modifications to correct the artifacts so that the view of the vegetation generated is realistic and acceptable.

Chapter 5
The modified Algorithm

In this chapter we present modifications to the 'slicing and blending' algorithm by Jauklin. The original algorithm produces visual artifacts. We modify the algorithm so that it displays all pre-rendered images simultaneously, and slicings are partitioned and rendered in a back-to-front order. This approach improves the quality of the stereo, maintains the basic appearance of the vegetation, and reduces visual artifacts; however, it increases rendering time slightly.

The modifications produce a rendering that is not totally faithful to the original vegetation model. Our goal is to produce acceptable stereo images of forests of vegetation in real-time and to produce views that appear "natural" in a walk-through environment. The images reproduced by our modified algorithm maintain the "treeness" property of vegetation, generating a close approximation of the original model.

Chapter four explained the problem introduced due to alpha blending. To solve this problem of parts appearing and disappearing, all the slicings are displayed, and alpha blending is eliminated. Initially we expected that when any slicing becomes parallel to viewing direction, the planar nature of the images would be easily seen. However, if a tree is sufficiently dense, this anomaly is not visible. Each of six slicings becomes parallel to the viewing direction once during a rotation around the tree, and this occurs only for 3 to 4 degrees for every slicing, which is only 2-3 frames. See Fig.10. The portions of vegetation inside the circle in figure 10(a) are approximately perpendicular to the viewer in figure 10(b).

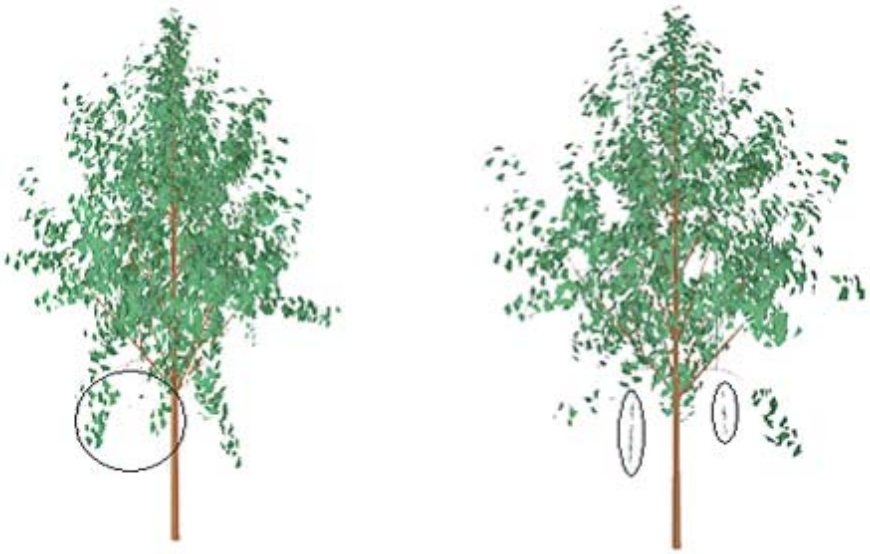


Fig. 14 (a) **Fig.14 (b)**

Figure 14 Nearly perpendicular slices.

Another disturbing anomaly in stereo is depth reversal. The parts of vegetation supposed to be behind are drawn over those that are meant to be in the front. This happens because the primary and secondary slicings intersect each other. Depth relations are reversed. Certain parts of vegetation appear to move in one direction and the remaining in the opposite direction. This occurs in stereo because depth is conveyed and can be very distracting.

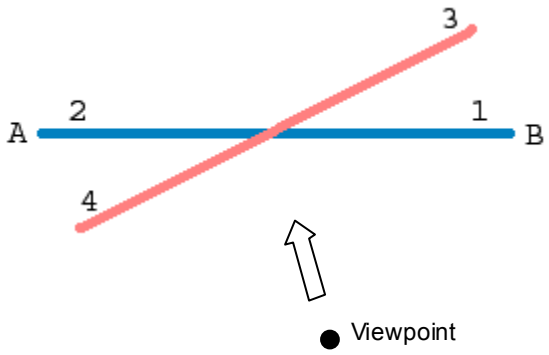


Figure 15 Depth Reversal

This phenomenon can be understood with the help of top view shown in Fig.11. Consider segments 1, 2, 3 and 4 to be drawn. Let 1 and 2 belong to line A and 3 and 4 belong to line B. If we draw A first followed by B, we observe that 3 lying behind 1 is drawn over 1. If the order of lines were reversed, 2 gets drawn on top of 4.

To correct this, the modified algorithm splits each slice vertically into two halves. Considering the pre-processing step, each plane is cut into two halves. Fig.12. illustrates this. We now refer to each half plane a slice. The left halves form one slicing and the right halves from another. From 3 viewpoints, distributed on a circle, 60 degrees apart and at same distance above ground, 6 such slicings are obtained. The center of this circle lies on a vertical axis V, through the centroid of the object (which is same for each slicing in our case). Other constraints remain the same as the original algorithm i.e., each slice has the same dimensions, and the same distance exists between adjacent slices. The coordinates in the scene to which each slice is to be mapped are pre-determined.

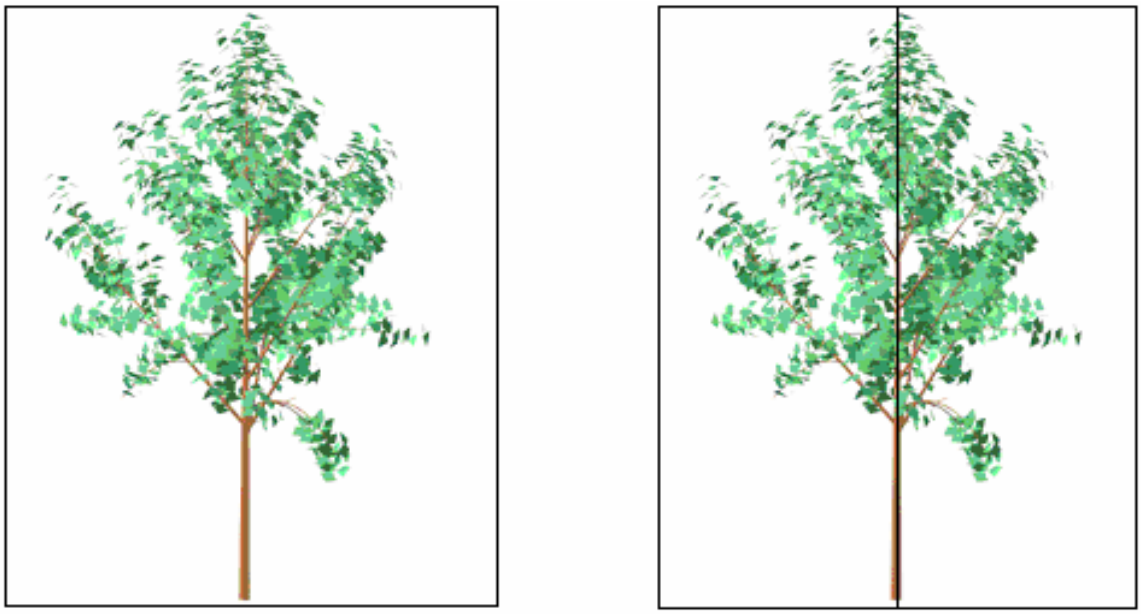


Fig. 16(a) **Fig. 16(b)**

Figure 16 Split each slice into two

The slicings are then displayed in back to front order. In addition, every slice of each slicing is drawn in back to front order relative to the viewpoint.

As shown in fig.13 below, we number slices from 1 to 6 in counterclockwise direction. To generate the view from point A, the order would be 2-1-3-6-4-5, and from point C it would be 6-5-1-4-2-3. Also, note, when drawing slicing 3 viewed from point C the order of slices would be e-d-c-b-a and from point A it would be a-b-c-d-e.

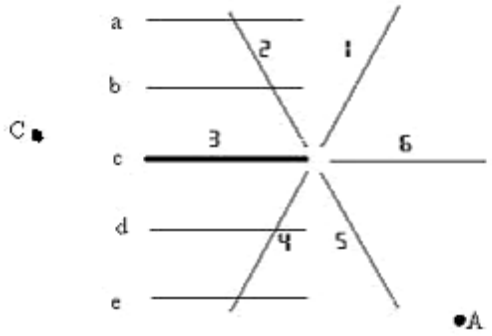


Figure 17 Order of rendering

The slicing and elimination of alpha blending produces a doubling effect. There are branches and parts of the vegetation that are present in two or three slicings. Since all slices are displayed, these parts would appear more than once. See fig.14 for the rendered tree using our modified algorithm and compare it with the original model in Fig.1. As mentioned in the introduction, our objective was to be able to render multiple trees in real-time in stereo producing images, which appear natural.

Figure 18 Vegetation using the modified algorithm

Another inherent defect in the Jauklin Algorithm is discontinuities in geometry created by the slicing. Different parts of a single branch may be projected on different planes of the same slicing. Fig.15. illustrates this. A branch with this property has been highlighted. Fortunately, this aberration usually goes unnoticed due to the dense nature and random complexity of vegetation. However, if this appears to be a problem, these parts of vegetation can be obscured by foliage.

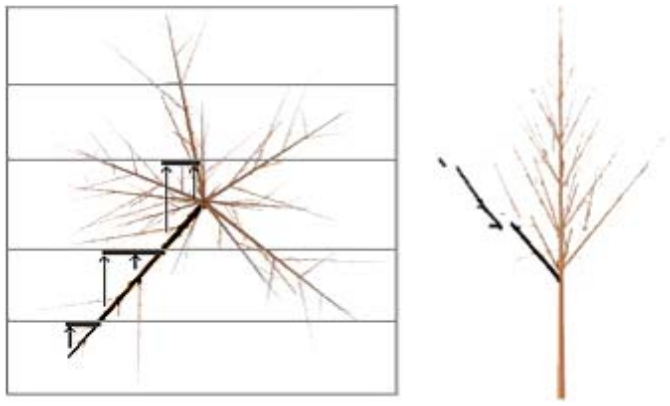


Figure 19 Discontinuities caused by the algorithm

Chapter 6
Implementation and Results

6.1 Pre-processing: Creation of slices

The 3-d model required to generate the slices was obtained from www.3dcafe.com. The model is a 3-D Max file of a tree. The model was rendered using the following parameter settings for illumination viz. ambient lighting and no directional light sources. The required clipping planes were then defined to represent the corresponding slices. Each slice was rendered and stored as an image file (256 x 256 or 512 x 512 pixels) image dimensions were restricted to powers of 2 for performance optimization. Every slice was then split over a vertical axis. The slices were created using an orthographic projection. The slices were taken by viewing the model from 3 view points 60 degrees apart. This gave us a total of 15 slices, each split into 2. If prior information regarding the position and orientation of the tree in the final scene is known, then it is possible to applying illumination specific to the tree before slicing it. However, for a generic case as in the example provided, the model had no special lighting.

6.2 Implementation

The algorithm was implemented in Visual C++ 6.0 using the OpenGL API. The slices generated in the preprocessing step were used as textures during implementation. OpenGL recommends the use of texture objects for this purpose. The OpenGL routine "glTexImage*D()" uses texture but needs to be

loaded every time it had to be rendered. A texture object stores the texture data and can be used whenever required. Due to the use of objects, the texture does not need to be loaded every time as the previously loaded data is used. Using such texture objects is the most efficient method for texturing. No blending was used to render the textures. The textures were loaded as RGBA pixels with each of the R, G and B values set to a value less than a threshold of 100. The alpha value is 1 giving the texture a 100 % transparency.
The same tree was also rendered using the Billboarding method mentioned in chapter 3.

6.3 System specifications

The code was run on a machine with an Intel Pentium 4 processor 1.7 GHz, 256 MB RAM, NVIDIA Quodro2 MXR/EX card and running a Windows 2000 operating system. The card provides Quad buffered stereo support. For machines without stereo capabilities, a separate view-port was defined for each eye and images were displayed for cross-eyed viewing.

6.4 Results

The following tables depict the rendering times using our algorithm and billboarding.

Resolution : 512 x 512	
Number of trees	Approximate Frames per sec.
1	38
2	22
3	19
4	14
6	11
8	8
10	6
15	4
20	3

Resolution : 256 x 256	
Number of trees	Approximate Frames per sec.
1	63
2	38
3	25
4	22
6	15
8	12
10	11
15	7
25	6

Table 1 Frame rate using the modified algorithm

Billboarding	
Number of trees	Approximate Frames per sec.
20	25
30	24
40	19
50	13
60	11
70	10
80	9
100	8
125	6
200	5
225	4

Table 2 Frame rate using Billboarding

Chapter 7
Discussion

7.1 Summary

In case of an environment where the viewer's attention is generally focused on a single 3-D model, it becomes necessary to be able to produce views of the model that change continuously as the user moves around in the environment. Doing this gives it the required realism to produce an immersive walk-through. The slicing method which we use gives us such capabilities.

In cases where a large amount of vegetation is present in an immersive scene, the attention of the viewer is not just focused on specific areas of the vegetation, but on the overall vegetation. This implies that there is no real need to render vegetation in detail. So the transparency, motion parallax and the limited vegetation complexity present in the scene can still be realistically rendered using billboarding techniques. In scenes where there is dense vegetation, the viewer is less likely to notice that he is being presented with the same view of a given tree or different views of the same tree. So billboarding is efficient in such situations and allows one to attain better frame rates, thus reducing the chance of flicker.

Slicing algorithm offers acceptable quality for vegetation at distance of about 10 to 50 meteres away. When the viewer is too close to the vegetation, one might consider using other representations offering better quality.

7.2 Future work

For both billboarding and slicing, the stereo rendering problem is easily partitionable in a multiple processor environment in a straightforward way. Hence, one would expect acceptable frame rates in this case, even with large numbers of trees in the scene. This is an area for future research.

The implementation does not support lighting and shadows. This aspect as regards to image based rendering has not yet been explored to a sufficient level.

We note that the current algorithms are not able to render correct views of the vegetation if the viewpoint lies above the highest point on the vegetation. Efficient stereo rendering of vegetation in real-time from arbitrary viewing points is still an outstanding problem. We seek solutions for this problem.

References

1. Jauklin, "Interactive Vegetation Rendering with Slicing and Blending," Eurographics 2000 short presentations.

2. D.F. McAllister (Editor), Stereo Computer Graphics and Other True 3D Technologies, p. 5,26, Princeton University Press, Princeton NJ, October 1993.

3. McReynolds, Tom, D. Blythe, B. Grantham and S. Nelson, “Programming with OpenGL: Advanced Techniques,” course 17 notes at SIGGRAPH’98, 1998.

4. S. E. Chen and L. Williams, "View interpolation for image synthesis," Proc. ACM SIGGRAPH 93, pp. 79-288, 1993.

5. N. Max and K. Ohsaki, "Rendering trees from pre-computed Z-buffer views," Eurographics Workshop on Rendering 1996, pp. 165-174, 1996.

6. S. E. Chen, “QuickTime VR – An Image-Based Approach to Virtual Environment Navigation”, SIGGRAPH 95, pp. 29 – 38, 1995.

7. J. Shade, S. Gortler, Li-wei He and R. Szeliski, "Layered Depth Images," SIGGRAPH ‘98, pp. 231-242, 1998.

8. G. Schaufler, "Per-Object Image Warping with Layered Impostors," Eurographics Rendering Workshop 1998, pp. 145-156.

9. G. Schaufler, "Image-based object representation by layered impostors," ACM symposium on Virtual Reality Software and Technology '98, pp. 99-104, 1998.

Appendix A

Stereo Images for cross-eyed viewing using Billboarding

Figures 20, 21 and 22 show a forest, from three pre-computed directions, at same height above ground and 30 degrees apart, rendered using the billboarding technique.

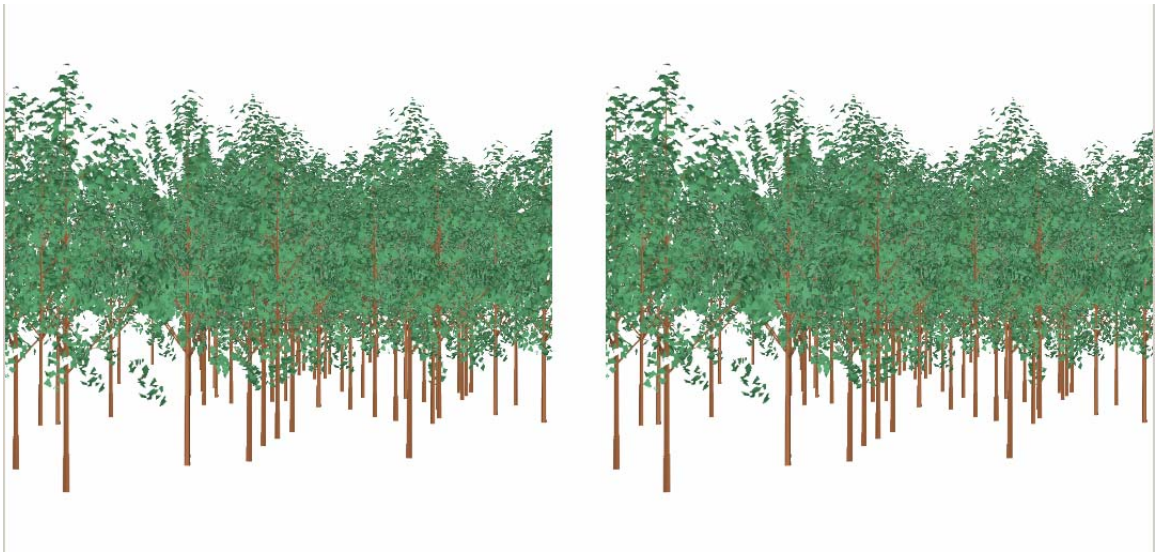


Figure 20

Figure 21

Figure 22

Appendix B

Stereo Images for cross-eyed viewing using Slicing

Figures 23, 24 and 25 show a tree rendered from three pre-computed directions, at same height above ground and 30 degrees apart using the billboarding technique.

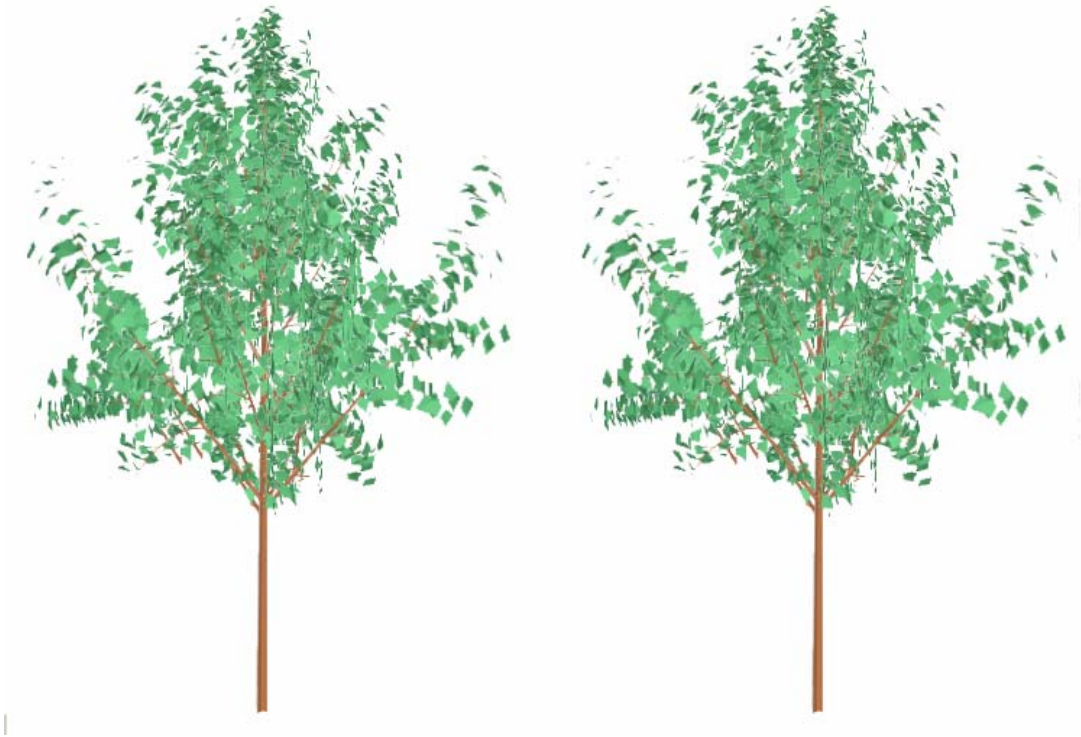


Figure 23

Figure 24

Figure 25