Generating Textures of New Zealand Native Wood

Jack Wang

Abstract - This report explores algorithms for computer generated textures simulating New Zealand native wood, we out line procedural texturing algorithms like Perlin noise and turbulence, techniques like texture generation, texture mapping and Ray tracing. The main goal of this research is to study New Zealand native wood in depth and to gather sufficient information to be able to generate realistic and detailed 3D texture for New Zealand native woods. In particular this work shows the development of texture for Kauri out of New Zealand's most famous native trees, and with Rimu.

1.Introduction

The goal of this research is to introduce and carry forward New Zealand native woods in the computer graphics field using 3D texture. There are many common and famous woods that have already been introduced in the computer graphics field as 3D textures such as Maple, Mahogany, Pine, and etc.[POV-RAY], and there are already many people using them to model wooden objects such as furniture, toy, wall and floor panels and architectures in computer-generated scenes frequently, and the same time finding them very easy and natural to the eyes, therefore I feel that it is a very interesting topic study more in depth, especially there are no 3D textures of New Zealand native woods published for people to use yet as far as I know, and yet the unique features and varieties of New Zealand does not lose its pride when comparing to other woods all around the world, I believe these beautiful and unique woods need to be promoted, and can be very popular in the future.

My goals of this research are to study the properties of New Zealand native woods (the wood grains, the colour, and the physical properties) as well as successfully generate detailed 3D texture of a particular wood that can be applied to objects and used to create a scene by a ray tracer.

Texture mapping, is a powerful technique for adding realism to a computer-generated scene. Firstly, why do I put my emphasis on "3D" texture mapping, why is it better? Let me give you an example, can you tell the difference between a solid block of wood and a veneered block of wood just by looking at it. You will probably say something like "Of course, you can tell them apart easily by looking at the edges to see if the grains are matching." that is exactly the reason, we do not just want to stick a texture on to the surface of an object like the conventional way, sometimes it is hard to match the textures side by side especially when the object is irregular and even if they do, the texture will most likely be distorted and lose its realism; this is where 3D texture comes in to solve problems like this.

2. Researched Areas

2.1. Texture Mapping

2.1.1. Contouring achieved by 1D texture mapping

Contour curves drawn on an object can provide valuable information about the object's geometry. Such curves may represent height above some plane (as in a topographic map) that is either fixed or moves with the object. Alternatively, the curves may indicate intrinsic surface properties, such as geodesics or loci of constant curvature.

Contouring is achieved with texture mapping by first defining a one-dimensional texture image that is of constant colour except at some spot along its length. Then, texture coordinates are computed for vertices of each polygon in the object to be contoured using a texture coordinate generation function. This function may calculate the distance of the vertex above some, or may depend on certain surface properties to produce, for instance, a curvature value. Modular arithmetic is used in texture coordinate interpolation to effectively cause the single linear texture image to repeat over and over. The result is lines across the polygons that comprise an object, leading to contour curves.

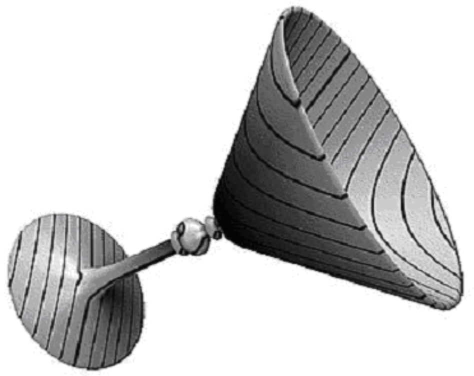


Figure 1: Showing distance from a plane with contouring [5]

2.1.2. 2D Texture mapping

The most common form of texture mapping is to apply a 2D image to a polygon (or some other surface facet) by assigning texture coordinates to the polygon's vertices. These coordinates index the texture image, and are interpolated across the polygon to determine, at each of the polygon's pixels, a texture image value. The result is that some portion of the texture image is mapped onto the polygon when the polygon is viewed on the screen. Typical two-dimensional images in this application are images of bricks or a road surface (in this case the texture image is often repeated across a polygon. Below is an example of "Wattie's Can" (2a) made from a 2D texture (2b) mapped onto a cylinder.

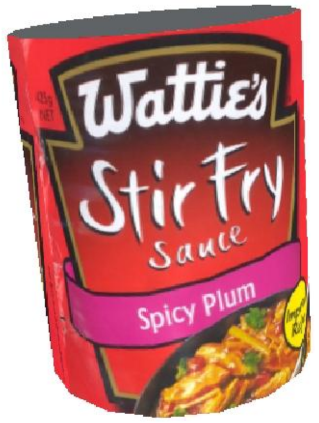


Figure 2a

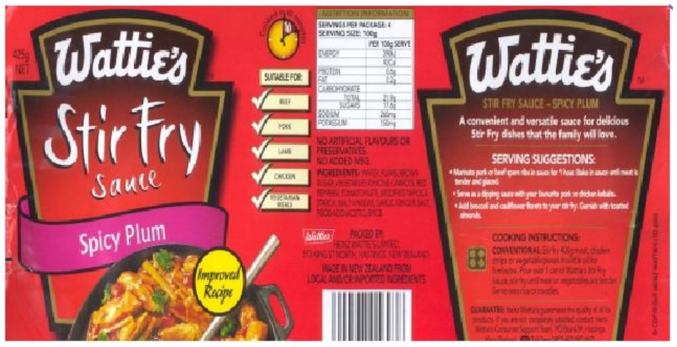


Figure 2b

Above images are from [2]

2.1.3. 3D Texture mapping

In 3D texture mapping, the volumetric data is copied into the 3D texture image. Then, slices perpendicular to the viewer are drawn. Each slice is again a texture mapped polygon, but this time the texture coordinates at the polygon's vertices determine a slice through the 3D texture image. This method requires a 3D texture mapping capability, but has the advantage that texture memory need be loaded only once no matter what the viewpoint. If the data are too numerous to fit in a single 3D image, the full volume may be rendered in multiple passes, placing only a portion of the volume data into the texture image on each pass. A three-dimensional image might represent a block of marble from which objects could be "sculpted." Below is a sample scene from POV-Ray made from 3D texture mapping, notice the grains on the sides are mating.

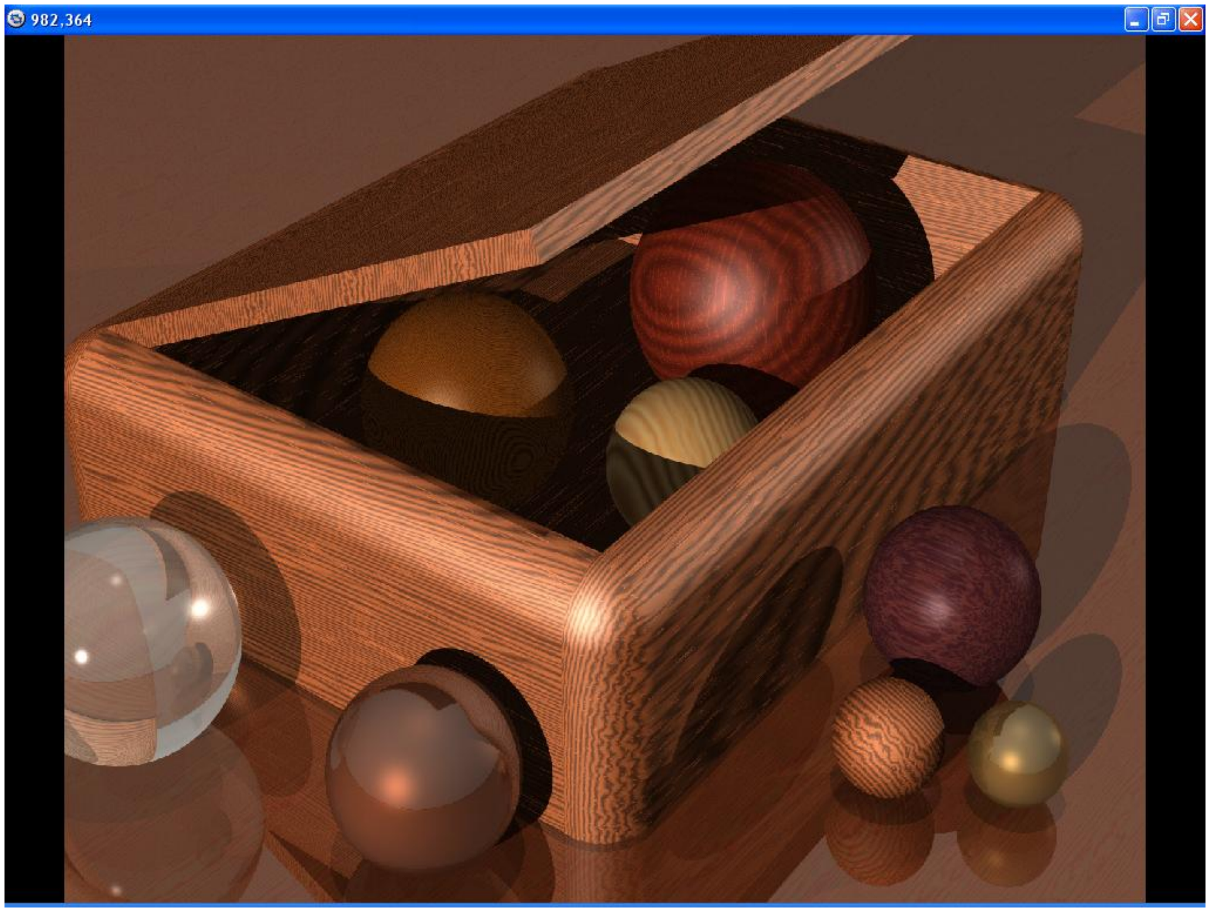


Figure 3: Wooden-box scene from POV-Ray

2.2. Rendering Texture Mapped Objects

2.2.1. Polygon Rendering

2.2.1.1. POV-Ray

During the process of my research, I came across a popular ray tracer program call POV-Ray which proved very useful for my project.

What is a Ray Tracer?

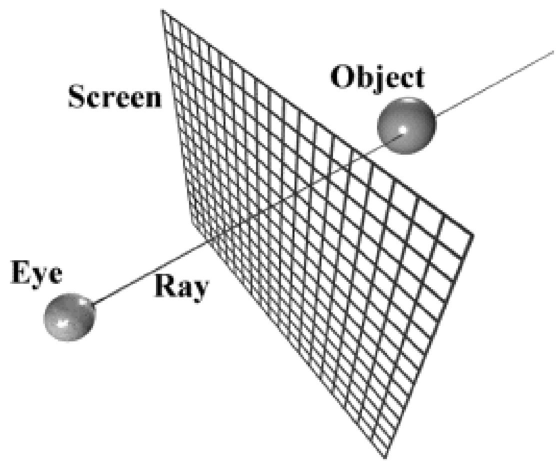


Figure 4: Logical Ray Tracing diagram [4]

Ray tracing is a method of generating realistic images by computer, in which the paths of individual rays of light are followed from the viewer to their points of origin. A ray tracer is any program that implements this method. Since ray tracing makes use of the actual physics and mathematics behind light, the images it produces can be strikingly life-like, or "photo-realistic."

POV-Ray is free and open for everybody to work on; most importantly, it provides good foundation in order to generate realistic scenes (Accurate lighting, material effects, good memory and performances techniques and so on.). I can take advantage off existing facilities and try to generate realistic 3D wooden textures. For my project, I will be working on generating material for Kauri.

2.3. Texture Generation

2.3.1. Perlin Noise

Many people have used random number generators in their programs to create unpredictability, make the motion and behavior of objects appear more natural, or generate textures. Random number generators certainly have their uses, but at times their output can be too harsh to appear natural. This article will present a function which has a very wide range of uses, more than I can think of, but basically anywhere where you need something to look natural in origin. If you look at many things in nature, you will notice that they are fractal. They have various levels of detail. A common example is the outline of a mountain range. It contains large variations in height (the mountains), medium variations (hills), small variations (boulders), tiny variations (stones) . . . you could go on. Look at almost anything: the distribution of patchy grass on a field, waves in the sea, the movements of an ant, the movement of branches of a tree, patterns in marble, and winds. All these phenomena exhibit the same pattern of large and small variations. The Perlin Noise function recreates this by simply adding up noisy functions at a range of different scales. To create a Perlin noise function, you will need two things, a Noise Function, and an Interpolation Function.

2.3.1.1. Noise Function

A noise function is essentially a seeded random number generator. It takes an integer as a parameter, and returns a random number based on that parameter. If you pass it the same parameter twice, it produces the same number twice. It is very important that it behaves in this way; otherwise the Perlin function will simply produce nonsense, a valid result from a Noise function can look like the graph below.

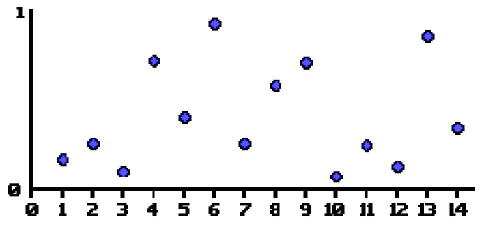


Figure 5: Noise Function [1]

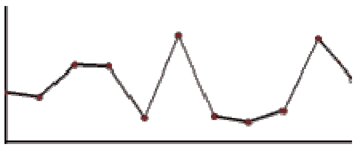
2.3.1.2. Interpolation Function

Once have created the noise function, you then need to smooth out the values it returns. Again, there are many methods, but some look better than others. A standard interpolation function takes three inputs, **a** and **b**, the values to be interpolated between, and **x** which takes a value between 0 and 1. The Interpolation function returns a value between **a** and **b** based on the value **x**. When **x** equals 0, it returns **a**, and when **x** is 1, it returns **b**. When **x** is between 0 and 1, it returns some value between **a** and **b**.

Linear Interpolation [1]:

This is the simplest and the fastest method, therefore the result is ugly. Unless you are trying to do Perlin noise in real-time, other methods are preferred.

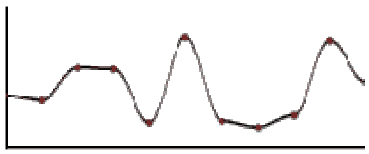
```
function Linear_Interpolate(a, b, x)
        return  a*(1-x) + b*x
        end of function
```



Cosine Interpolation [1]:

This method gives a much smother curve than Linear Interpolation. It's clearly better and worth the effort if you can afford the very slight loss in speed.

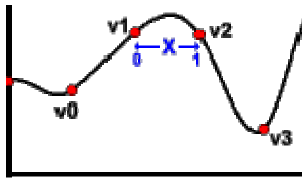
```
function Cosine_Interpolate(a, b, x)
      ft = x * 3.1415927
      f = (1 - cos(ft)) * .5

      return  a*(1-f) + b*f
end of function
```



Cubic Interpolation [1]:

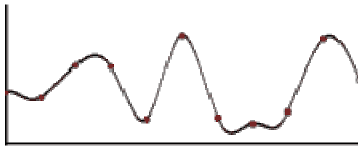
This method produces very smooth results indeed, but you will pay for it in speed. Actually, it would not give noticeably better results than Cosine Interpolation, but here it is anyway if you want it. It's a little more complicated, so pay attention. Whereas before, the interpolation functions took three inputs, the cubic interpolation takes five. Instead of just **a** and **b**, you now need **v0**, **v1**, **v2** and **v3**, along with **x** as before. These are:

v0 = the point before **a**
v1 = the point **a**
v2 = the point **b**



v3 = the point after **b**

```
function Cubic_Interpolate(v0, v1, v2, v3,x)
      P = (v3 - v2) - (v0 - v1)
      Q = (v0 - v1) - P
      R = v2 - v0
      S = v1

      return $Px^3 + Qx^2 + Rx + S$
end of function
```



The Perlin Noise therefore can use a cubic polynomial function interpolation to generate pseudo-random number at every point in the texture space, generates images.

Below is a sample image from POV-Ray that is able to be generated by Noise. As we can see, there are textures of marble and wood, they look realistic through the semi-randomness. Above section and diagram summarizes results from [1].

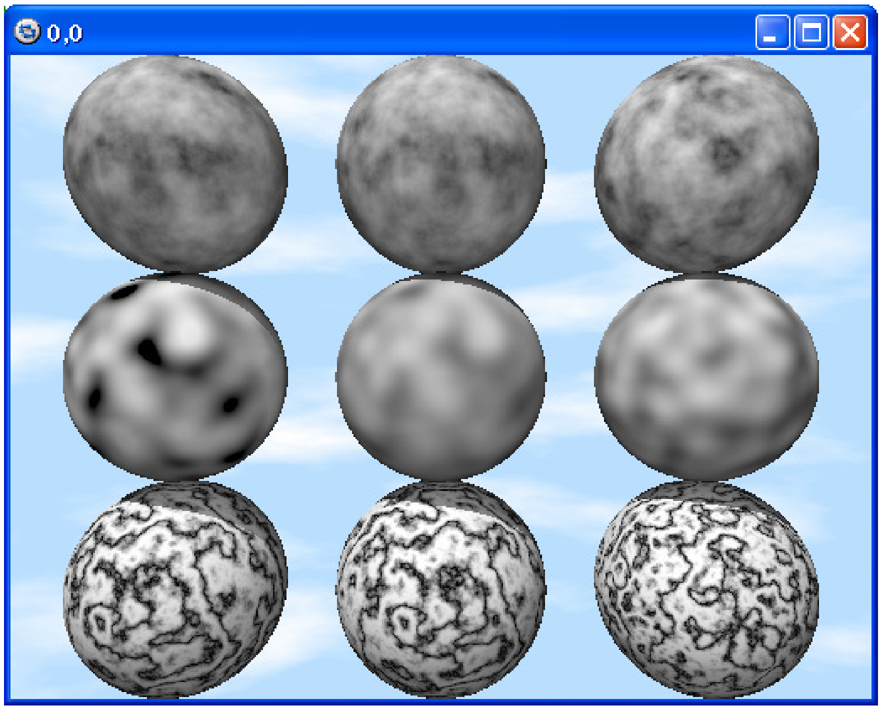


Figure 6: Marble scene from POV-Ray

2.4. Turbulence

"Noise is Good, Turbulence is better" [4]

We can impose self-similarity on noise to add scale-invariance. The turbulence at a point is created by summing the noise at that point with scaled down noise values at other points.

$$\text{turbulence}(x) = \sum_{i=0}^{k} \text{abs}(\text{noise}(2^i x) / 2^i)$$

Where k is the smallest integer for which $\frac{1}{2^k+1}$ is greater than the size of a pixel in real coordinates. This is very similar to fractal surface generation, giving a visual impression of Brownian motion.

Here is an example of a turbulent field using the random field as a basis, with grey scale and random colour map.

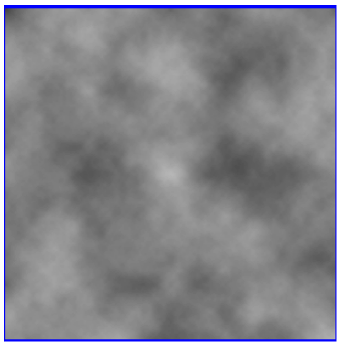


Figure 7a: Grey scale [4]

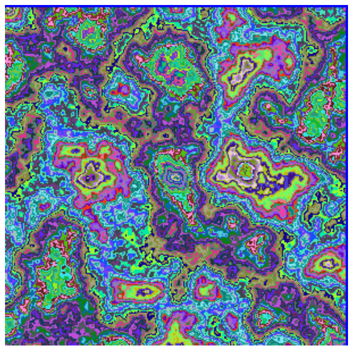


Figure 7b: Random Color [4]

Using Turbulence

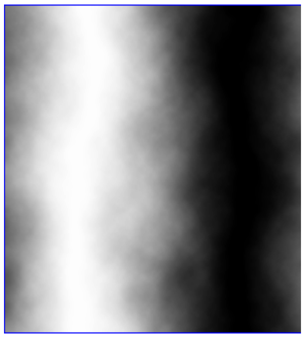
1. Start with a simple regular structure (e.g. a curve)
2. Impose turbulence on values

   $V(x) = V(x) + tubulence(x)$ or $V(x) = V(x + tubulence(x))$ [4]

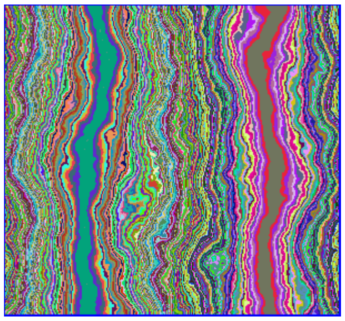
3. Need to modify range of turbulence to fit structure
4. Need to play with degree and scale of turbulence

Applications [4]

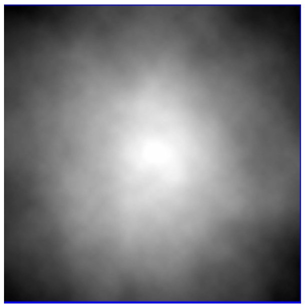
***Marble Texture**:* Add turbulence to sine wave, use a curve through colour space instead of a ramp. [4]



Here is an example of a sine wave with added turbulence. I show it using grey scale and a random colour map. [4]



***Stars**:* Intensity of points based on distance to the center. Add turbulence to this distance. Here is an example of a star with added turbulence. [4]



***Flames**:* Compute intensity of point based on distance from center in x. Scale it based on distance in y. Add turbulence. Use 3-D turbulence to animate. Here is an example of a flame with added turbulence. [4]

- Lots of interesting effects can be gained by adding turbulence
- Need to play with degree and scale to get most realistic images

- Ties together a lot of topics in graphics (fractals, texture, colour, curves)

Above section summarizes results from [4], please refer to it for more information on turbulence.

3. Examination of Wood

3.1. Brief introduction to Kauri

Kauri is an endemic conifer pine found in the northern third of the North Island of New Zealand. They grow from sea level to 600m in altitude. Kauri (*Agathis Australis*) belong to the family *Araucariaceae*. They are the only species of this very ancient family found in New Zealand. Other examples are found in the South-western pacific including Australia and New Guinea.

For the first 50 years of a kauri tree's life it is in a cone shape. From then on a solid trunk forms with a crown of branches from above 15 - 20m. They reach maturity at around 200 years but are still growing at 500 years old.

Figure 8: Just over 50 years old Kauri [6]

Kauri is one of the few native trees that are non-flowering. They are a type of gymnosperm (have a naked seed) with the seeds found in cones. The female cones are round in shape and the male cones are finger-shaped. The female cones mature in March and the seeds are quick to germinate as long as there is good light. Male and female cones are found on the same tree and they can be self-fertilized or the wind can blow the pollen from another tree. The seeds are also spread via the wind.

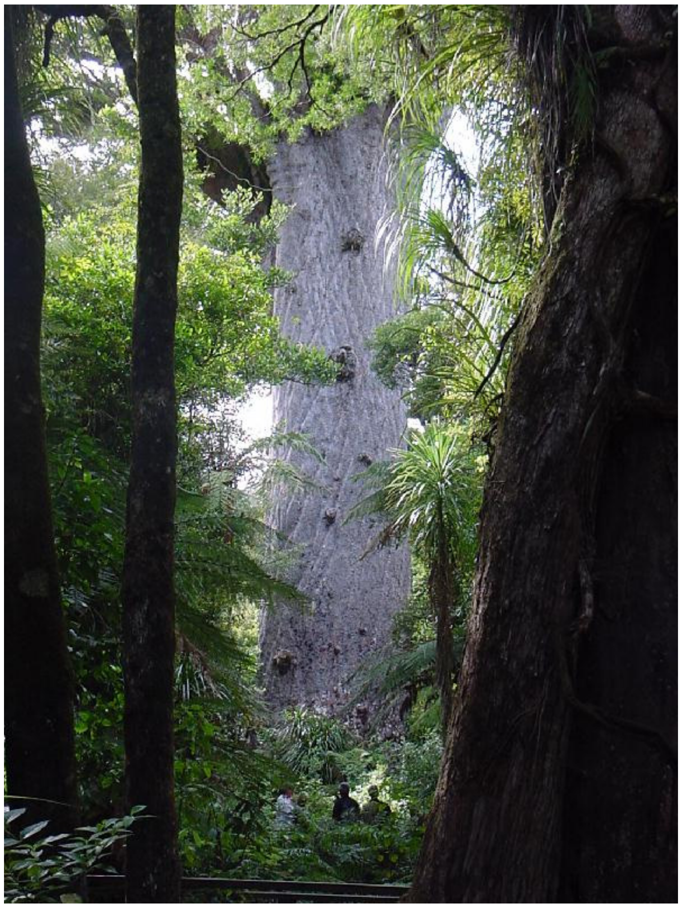


Figure 9: A grown Kauri tree [Google]

The trunk of a kauri tree h as grey coloured bark which flakes off in round pieces leaving flat marks - like hammer marks. Occasionally on the trunk there will be fresh gum bleeding from the tree.

The kauri is a host tree to many shrubs and tufting plants that live high up in the crown of the tree.

Kauri trees grows steadily throughout winter and summer, causes having more smooth and even grains and since they contain rich in gum (rosin), the texture on the polished surface has rich sheen and sometimes wavy luster, which are popular for interior building and furniture making. [6]

3.2. My own experiments

With the help of my old college school teacher, I was able to scan slices of a Kauri block by using the Buzzer facility to take off one millimeter at a time from the block, then I could used a scanner to scan the surface into my computer.

Figure 10&11: Mr. Bordot and me at the workshop process and scan Kauri

A total of 90 good quality images are what I capture from a 100*100*180mm block, each of them is 1.5MBs, I first study them, and from the study of the images I've found the colour range, the different grains it is composed of and many more useful data to help developing the 3D textures, more details will be mentioned later on.

An idea was suggested by my supervisor; it would be interesting to test if hue value from all colour on the scanned images of wood are the same, a surprising find was that the hue values are only between 4 and 15 out of 255, which is considered to be a relative small range; this is very useful information when choosing colour to generate 3D texture later on. I would imagine the reason for this is because the trunk of the tree is grown out of the same cells, therefore it's only the matter of the speed the tree is growing relative to the size of the cells and the density of the wood, and only the saturation changes, above was the reason I assumed that causes this effect. (Note, theoretically, I believe all the colours from the same wood material can be from one hue, and why don't the results show that? I believe it is because first, the Buzzer can not guarantee the smoothness of the surface if the planning is against the grain, fibers on the rough surface can cause inaccurate scanning results. And other problem is with devices' gamut can have unavoidable errors, and this is a common problem known in the computer graphics world.)

4. Implementation

4.1. Initial tests

Here are some screen shots of some 3D textures that I made manually just to try it out, to get familiar with work with 3D textures. I do this by using the Ray tracer assignment used in the 372(Computer Graphics) course.

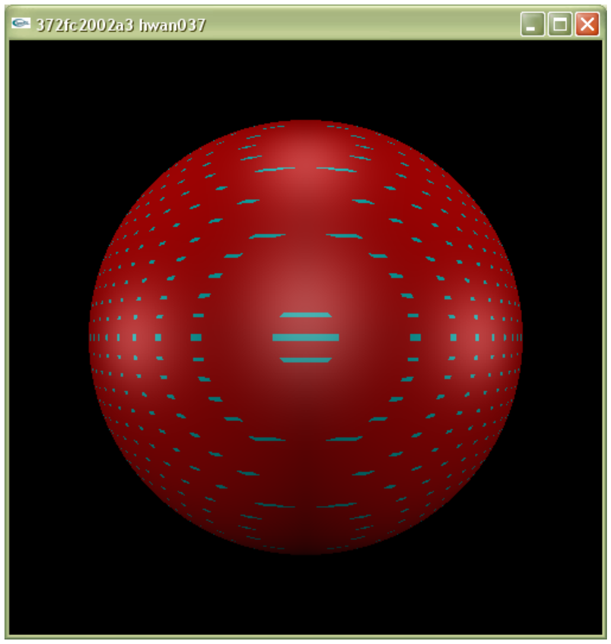


Figure 12: A solid red sphere with horizontal blue strips running through it.

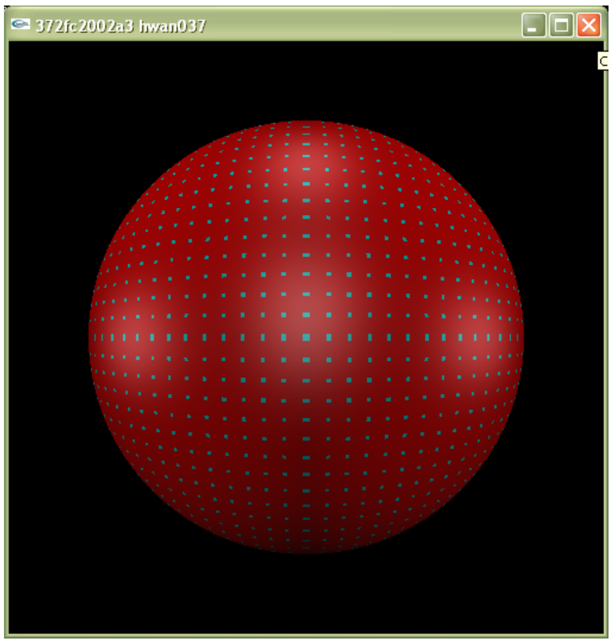


Figure 13: A solid red sphere with little blue cubes evenly distributed in it.

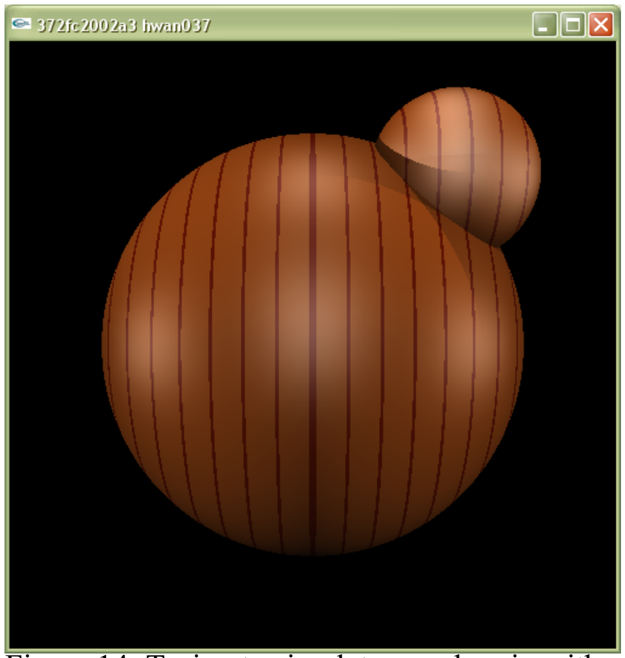


Figure 14: Trying to simulate wood grain with Light and dark brown layers.

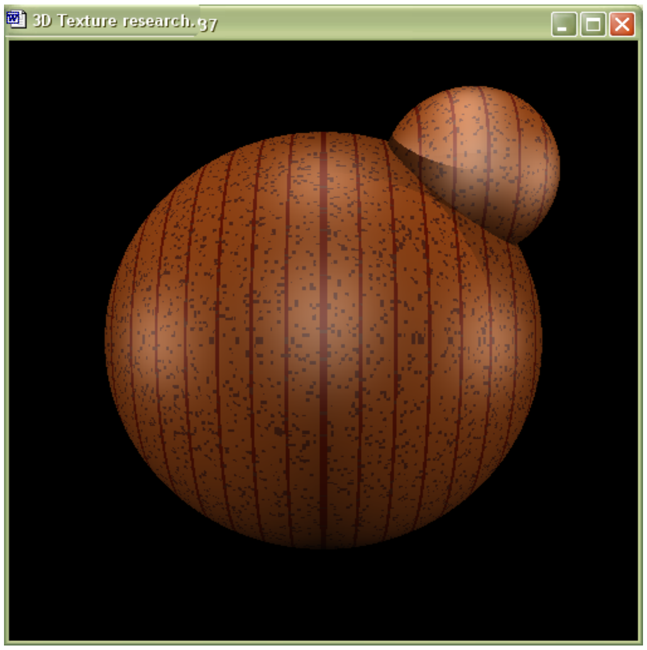


Figure 15: Same as 14 with random dark brown dots.

4.1.1. Problems Found and Solutions

Immediately I find that I run out of memory very easily, therefore I thought it was nearly impossible to produce very fine textures (200*200*200)

To solve this problem, I have to use bytes instead of floats for each of the RGB values to save memory to produce better resolution images; therefore the colour quality is compromised.

By using the 90 scanned kauri images(Figure 16) I can produce images like below (Figure 17).
As we can see, the wood grain of kauri is not very distinctive, as is it grown steadily during either winter or summer. Somehow the image came out with more red than I expected. My supervisor suggested that this could be an aliasing problem, this problem is caused by having a much higher resolution texture comparing to the pixel resolution on the monitor, and during the process of ray tracing, part of the texture colour is skipped, and therefore causes inaccurate result. On the other hand, if the texture resolution is much lower than the pixel resolution, then this is not good neither, as many pixels would have the same colour from the texture, therefore produced blocky appearance.
Now that I have the product from my scanning results, and found out that by doing it this way is not particularly useful, because of the sluggish image quality and the huge memory consumption, I realized that this way is not going to work; I will need to find another different approach.

Here is the C++ code for generating the 3D texture…

```
ifstream textureFile;
char* fileName="0.ppm";
textureFile.open(fileName, ios::in);
if (textureFile.fail())
        displayMessage(ERROR_MESSAGE, "could not open file %s",fileName);
skipLine(textureFile);
skipLine(textureFile);
textureFile >> textureWidth;
textureFile >> textureHeight;
textureDepth = 400;
cout<<"Height = "<<textureHeight;
cout<<"Width = "<<textureWidth;
textureFile >> numRGBValues;
```

```
int i,k,l,c,r,g,b;
for(i=1;i<textureDepth+1;i++){
        for(k=0;k<textureHeight;k++)
                for(l=0;l<textureWidth;l++){
                        textureFile >> r;
                        textureFile >> g;
                        textureFile >> b;
                        texture[k][l][i] = Color3(((float)r)/255,((float)g)/255,((float)b)/255);
                }
        string fileNumber = i.toString();
        char* fileName=fileNumber+".ppm";
        textureFile.open(fileName, ios::in);
        if (textureFile.fail())
                displayMessage(ERROR_MESSAGE, "could not open file %s",fileName);
        skipLine(textureFile);
        skipLine(textureFile);
        textureFile >> textureWidth;
        textureFile >> textureHeight;
        textureFile >> numRGBValues;
        textureDepth = 90;
}
```

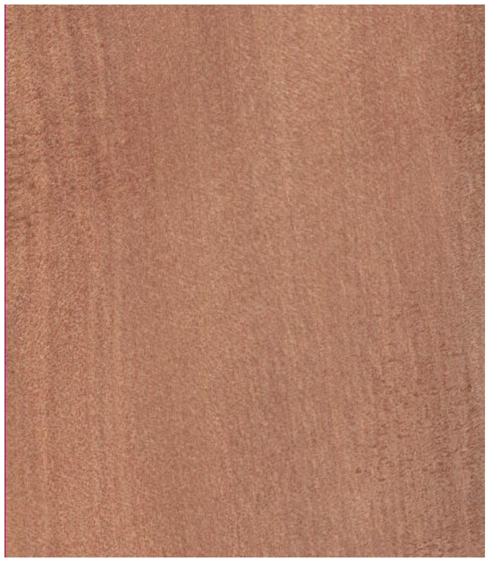


Figure 16: One of the scanned image from the Kauri slices

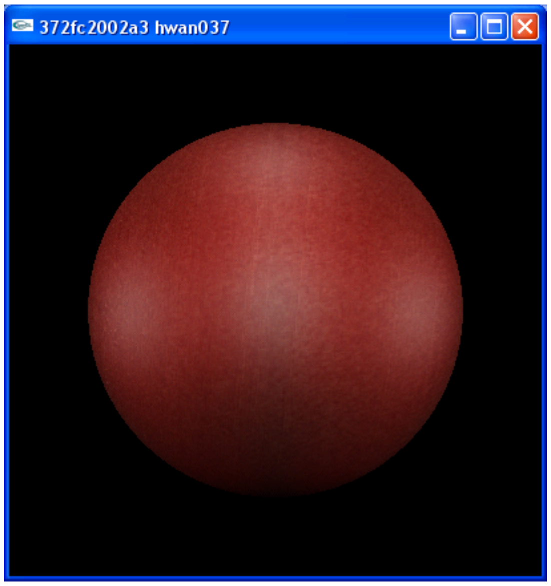


Figure 17: Kauri Sphere from a raw 3D texture

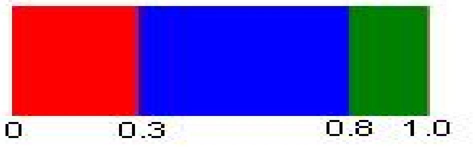
From this approach, the biggest problem is that the resolution is fixed,

4.2. Texture Generation

After my unsuccessful attempt, I have looked at other people's approach to wood texture generation. Here is how POV-Ray supports wooden textures.

4.2.1. How others generate realistic wooden textures

In POV-Ray, first we will have to declare colour_maps for how your texture colour is arranged from 0 to 1. E.g. From 0 to 0.3 is Red, 0.3 to 0.8 is Blue, and from 0.8 to 1.0 to Green.
You will form the following colour map



Secondly, we can declare a texture by combining a texture or more textures together to form layers, in each texture, we can specify the pigment, which is the way you want your colour_map to arrange, in this case, we use wood. If we choose wood the colour_map will be arranged like the picture below.

To create a wood texture, first you need to generate a cylinder simulating the trunk of a tree. (Figure 18)

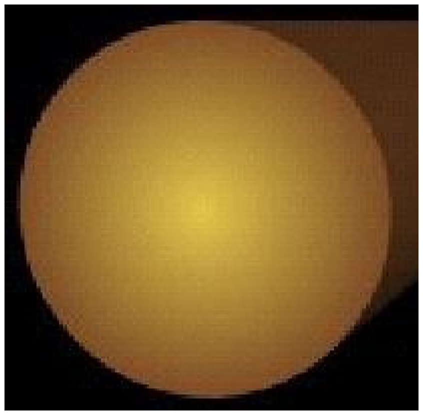


Figure 18:

And then add in the wood grain into the cylinder, (Figure19) the code follows..

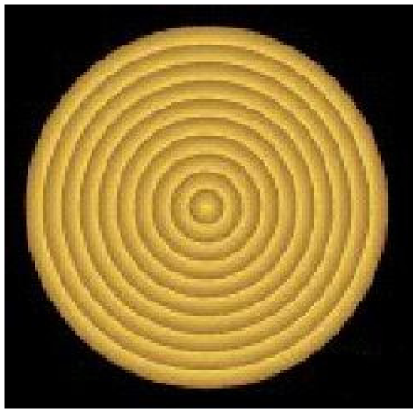


Figure 19:

```
#declare DemoWood=texture {
  pigment {
    wood              //Key word for having the texture circular like the grains
    color_map {
      [0, 1 color rgb <0.90, 0.80, 0.30>   //First coluor
             color rgb <0.50, 0.30, 0.15>]  //Second colour
    }
    ramp_wave      // Means the color varies like a ramp(first colour to second colour)
  }
}
#declare Log=cylinder{ <0, 0, 0>, <0, 0, 10>, 1 texture { DemoWood } }
// Create a cylinder with radius 1 and height of 10 and apply the texture,
there for we have only one year ring.
or
#declare Log=cylinder{ <0, 0, 0>, <0, 0, 100>, 10 texture { DemoWood } }
// Create a cylinder with radius 10 and height of 100 and will need to scale
the object down 10 time to see it. Now we would have 10 year rings, each
ring repeats the color_map that is given to it.
```

The cut-open view of the cylinder,

Unfortunately real wood doesn't look like this. (Figure 20)

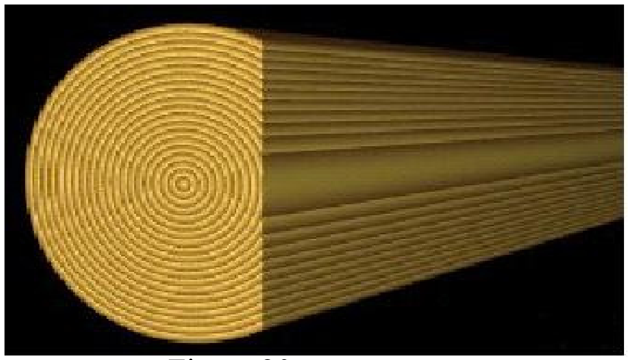


Figure 20:

It looks more like this. Why? It is because a tree trunk is not cylindrical; it is slightly conical, thicker at the root and narrower towards the crown. A cut parallel to the core will therefore be at a slight angle to the grain, resulting in the typical parabolic pattern. Code follows…

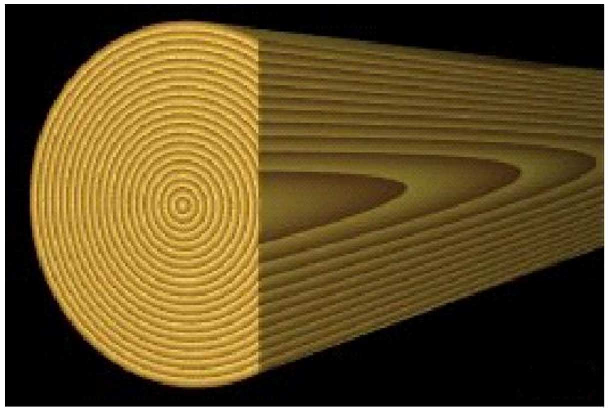


Figure 21:

```
#declare Log=difference {
    cylinder { <0, 0, 0>, <0, 0, 250>, 25

                  rotate <0, -1, 0> //Rotate the cylinder by 1 degree

    }

    plane {-x, 0 translate <4.6, 0, 0> }

    texture { DemoWood

                rotate <0, -1, 0> //Rotate the texture by 1 degree as well

    }

}
```

This section summarizes results from [3]

5.Results

5.1. Final Product

5.1.1. Kauri

This is what I have finally come up with using POV-Ray.
I have combined three different 3D textures together to form the final product; each of them models different characteristics of the Kauri. Texture 1 models the year ring of Kauri; they are unobvious, straight and evenly spaced. (Figure 22)

Texture 1:

```
#declare M_KauriA(Year Ring) =
colour_map {
   [0.00 0.10 color rgb < 0.59, 0.42, 0.12 >
           color rgb < 0.58, 0.43, 0.11 >]
   [0.10 0.80 color rgb < 0.62, 0.44, 0.12 >
           color rgb < 0.64, 0.49, 0.14 >]
   [0.80 1.00 color rgb < 0.63, 0.48, 0.13 >
           color rgb < 0.69, 0.42, 0.12 >]
}
```

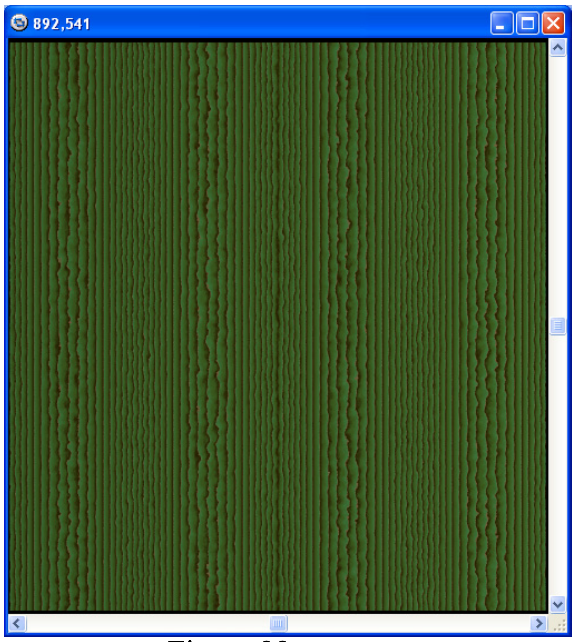


Figure 22:

Texture 2 models the brown dots between the grains, and they are formed by the fine fiber-like lines going cross the year rings. (Figure 23)

Texture 2:

```
#declare M_KauriB(Cross Grain) =
colour_map {
[0.00 0.30 color rgb < 0.45, 0.346, 0.04,0.85>
           color rgb < 0.58, 0.456, 0.04,0.85 >]
  [0.30 0.70 color rgb < 0.62, 0.454, 0.04,0.9 >
           color rgb < 0.63, 0.484, 0.040,0.85 >]
  [0.70 0.95 color rgb < 0.860, 0.584, 0.040,0.85 >
           color rgb < 0.860, 0.584, 0.040,0.9 >]
  [0.95 1.00 color rgb < 0.694, 0.524, 0.040,0.85 >
           color rgb < 0.714, 0.524, 0.040,0.85 >]
 }
```

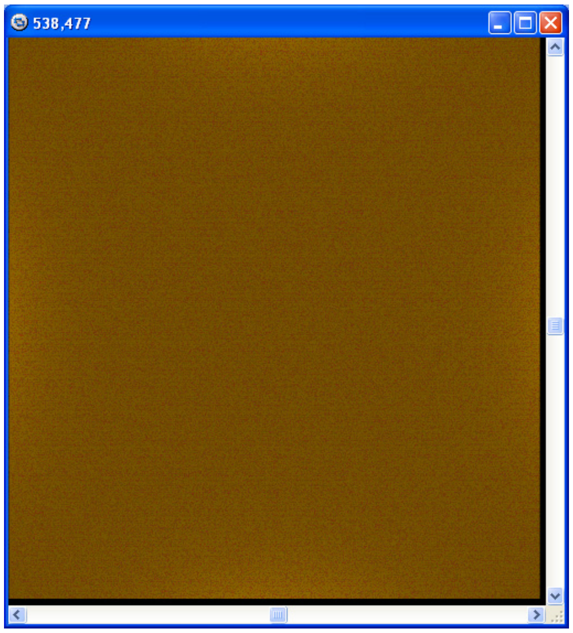


Figure 23:

Texture 3 models the wavy reflection when shined under light; this is a very nice characteristic of Kauri, the effect is produce by the transparent colours in the 3D texture. (Figure 24)

Texture 3:

```
#declare M_KauriC(Transparent Shine) =
colour_map {
  [0.00 0.30 colour rgbt < 0.40, 0.100, 0.215, 0.8 >
           colour rgbt < 0.50, 0.125, 0.225, 0.8 >]
  [0.30 0.50 colour rgbt < 0.50, 0.125, 0.225, 0.8 >
           colour rgbt < 0.75, 0.135, 0.230, 0.8 >]
  [0.50 0.70 colour rgbt < 0.75, 0.135, 0.235, 0.8 >
           colour rgbt < 0.50, 0.120, 0.225, 0.8 >]
  [0.70 1.0  colour rgbt < 0.50, 0.120, 0.225, 0.8 >
           colour rgbt < 0.40, 0.100, 0.215, 0.8 >]
}
```

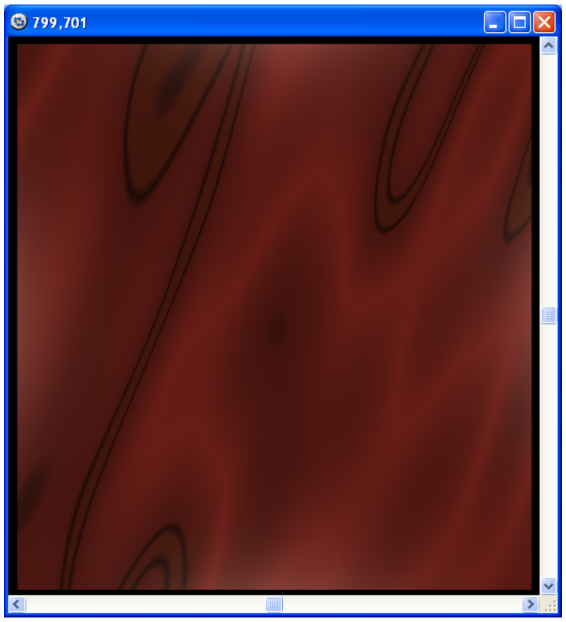


Figure 24:

By adding the 3 textures together, the texture of Kauri is produced in Texture 4. (Figure 25)

Texture 4:

The omega value controls how large each successive octave step is compared to the previous value. Each successive octave of turbulence is multiplied by the omega value. The default omega 0.5 means that each octave is 1/2 the size of the previous one. Higher omega values mean that 2nd, 3rd, 4th and up octaves contribute more turbulence giving a sharper, *crinkly* look while smaller omegas give a fuzzy kind of turbulence that gets blurry in places.

The octaves keyword may be followed by an integer value to control the number of steps of turbulence that are computed. Legal values range from 1 to <10. The default value of 6 is a fairly high value; you won't see much change by setting it to a higher value because the extra steps are too small. Float values are truncated to integer. Smaller numbers of octaves give a gentler, wavy turbulence and computes faster. Higher octaves create more jagged or fuzzy turbulence and takes longer to compute.

```
#declare T_KauriWood  =
texture {       pigment {
                wood
                turbulence 0.02
                octave 10
                omega 1.15
                scale <0.13, 0.13, 0.15>
                colour_map {
                        M_KauriA(Year Ring)
                }
                ramp_wave
        }
}
texture {       pigment {
                wood
                turbulence <0.2, 0.2, 1>
                scale <0.03, 0.03, 0.03>
                translate x*14
                colour_map {
                        M_KauriB(Cross Grain)
                }
                ramp_wave
                }
        }
texture {
    pigment { wood
        colour_map {
            M_KauriC(Transparent Shine)
        }
        omega 0.8
        turbulence <0.01, 0.01, 10>
        scale <17.5, 5.5, 10>
        translate x*13
        rotate<0,20,0>
```

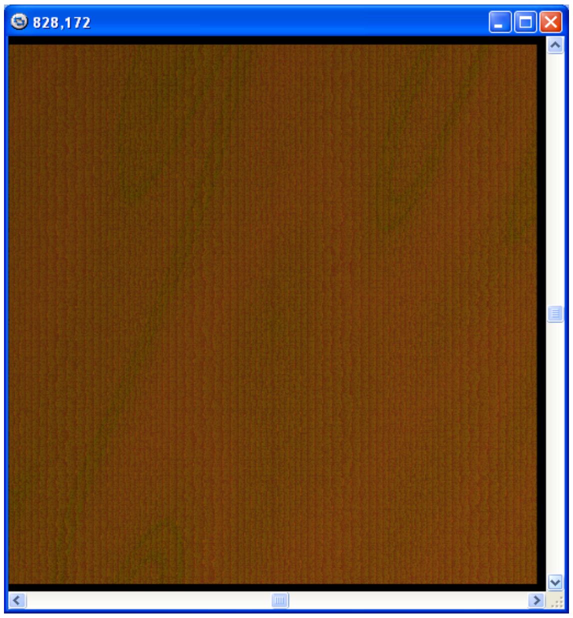


Figure 25:

```
    }
}
```

Lastly, we can apply the texture to a box just for example to produce the following result. (Figure 26)

```
box {<-3.75, 0.0, -2.75> <3.75, 2, 2.75>
      rotate <50, 0, 0>
      scale <2,2,2>
      texture {
            T_KauriWood
            rotate <50, 0, 00>
            scale <2,2,2>
            translate x*12
      }
}
```

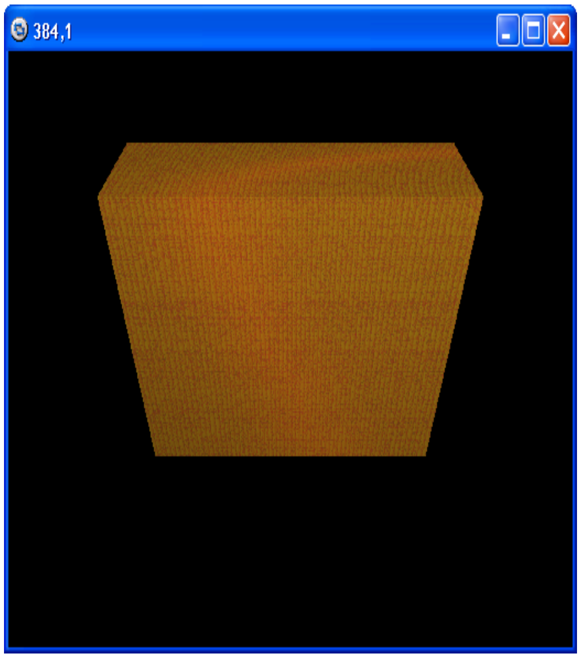


Figure 26:

5.1.2. Rimu

Same with Kauri, but this time, we only need 2 layers, without reflective layer. Texture1 models the color and the grain. (Figure 27)

Texture 1:

```
  #declare M_RimuA =
  colour_map {
     [0.00 0.04 color rgb < 0.30, 0.205, 0.075>
               color rgb < 0.40, 0.205, 0.075 >]
     [0.04 1.00 color rgb < 0.40, 0.212, 0.090 >
               color rgb < 0.50, 0.225, 0.095 >]
  }
```

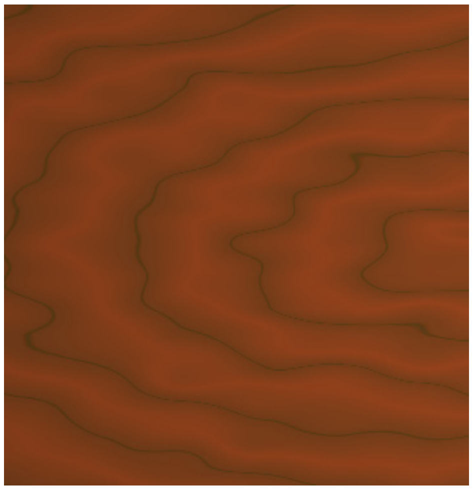


Figure 27:

Texture2 models the light-colored pores. (Figure 28)

Texture 2:

```
  #declare M_RimuB =
  colour_map {
     [0.00 0.10 color rgbt < 0.50, 0.172, 0.115, 0.5 >
               color rgbt < 0.50, 0.172, 0.115, 0.7 >]
     [0.10 0.15 color rgbt < 0.50, 0.172, 0.115, 0.7 >
               color rgbt < 0.35, 0.115, 0.060, 0.9 >]
     [0.15 0.20 color rgbt < 0.50, 0.172, 0.115, 0.9 >
               color rgbt < 0.35, 0.115, 0.060, 0.7 >]
     [0.20 1.0  color rgbt < 0.35, 0.115, 0.060, 0.7 >
               color rgbt < 0.35, 0.115, 0.060, 0.5 >]
  }
```

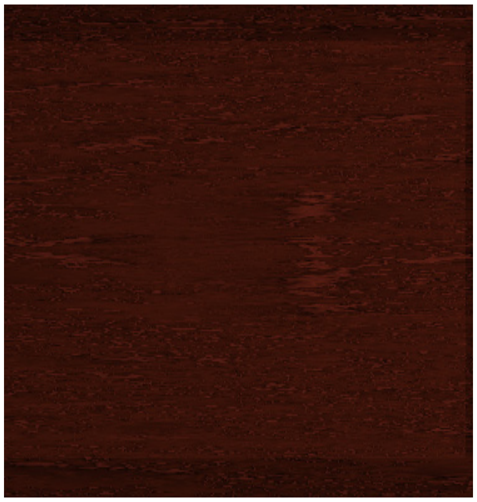


Figure 28:

Texture3 is the combined effect. (Figure 29)

Texture 3:
```
        #declare **T_RimuWood** =
          texture {
              pigment{
                  wood
                  colour_map { **M_RimuA** }
                  turbulence <0, 0.6, 0>
                  omega 2
                  turbulence 0.045
                  scale <2, 0.7, 0.5>
                  translate < -3, 0, 0 >
                  rotate <-3, -3, 0>
              }
          }
          texture {
              pigment{
                  wood
                  colour_map { **M_RimuB** }
                  turbulence <0, 0.7, 0>
                  omega 0.7
                  scale <5, 0.175, 1>
                  rotate x*70
              }
          }
```

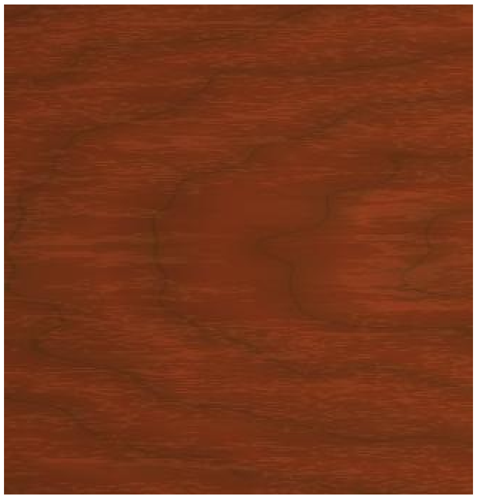


Figure 29:

And here is a scene with Rimu texture applied to a box. (Figure 30)

```
box {<-3.75, 0.0, -2.75> <3.75, 2, 2.75>
      rotate <50, 0, 0>
      scale <2,2,2>
      texture {
            **T_RimuWood**
            rotate <50, 0, 00>
            scale <2,2,2>
            translate x*12
      }
}
```

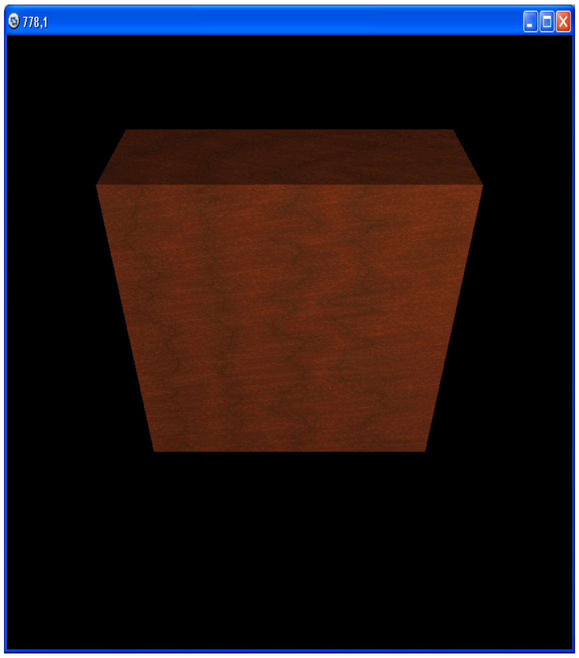


Figure 30:

5.2. Sample Scenes For Kauri using POV-Ray

5.2.1. Kauri

From the texture created, I am able to generate scenes like below. Below is a scene simulates a block of Kauri. As you can see on the block, the fine straight grains and the reflective layer from the light are very distinctively shown. (Figure 31)

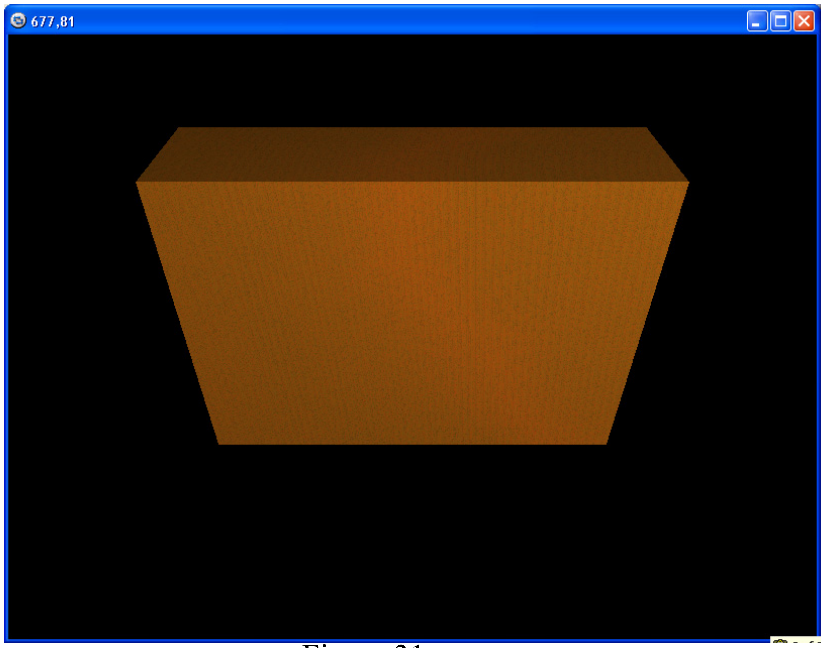


Figure 31:

Below is a scene with the same block but only applied with of the transparent layer, to show how the reflective effect is achieved. (Figure 32)

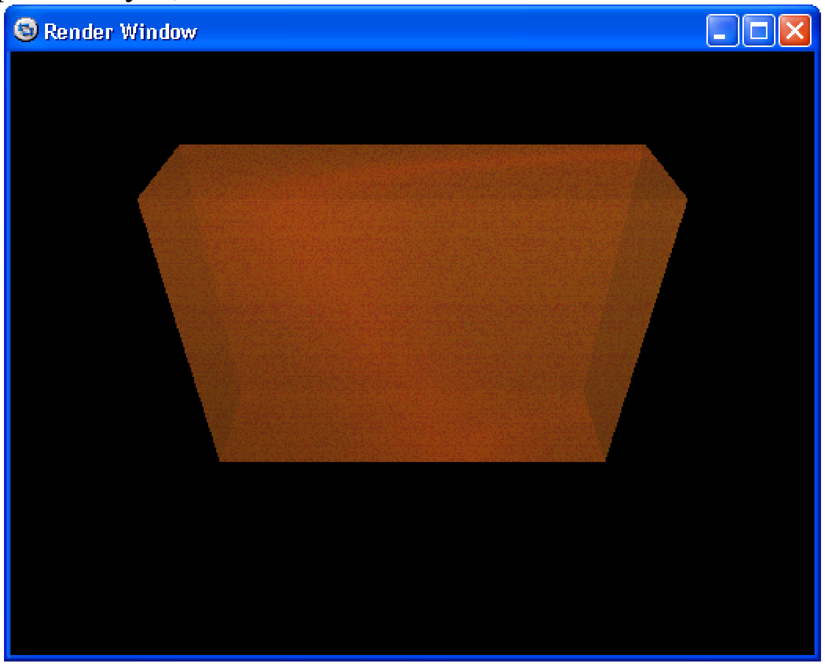


Figure 32:

Below is another example, like a wooden sculpture done by the lathe. (Figure 33)

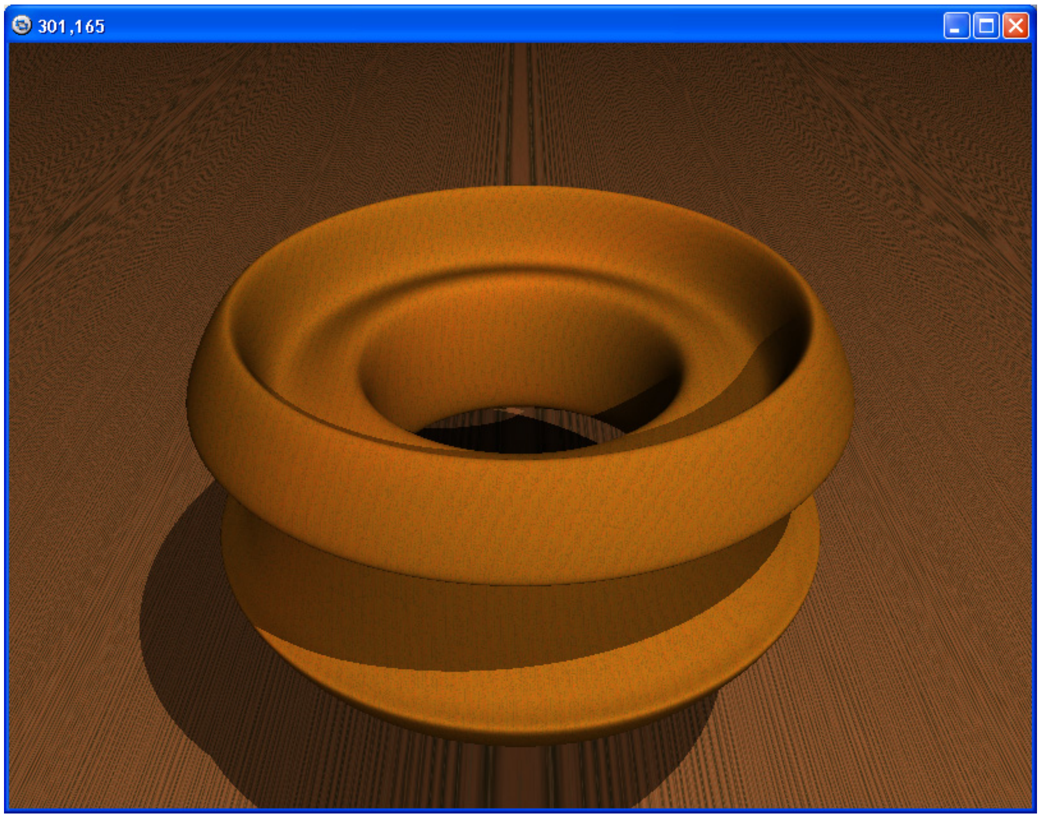


Figure 33:

Below is a scene with 3 Kauri-made pawns on a wooden chessboard. (Figure 34)

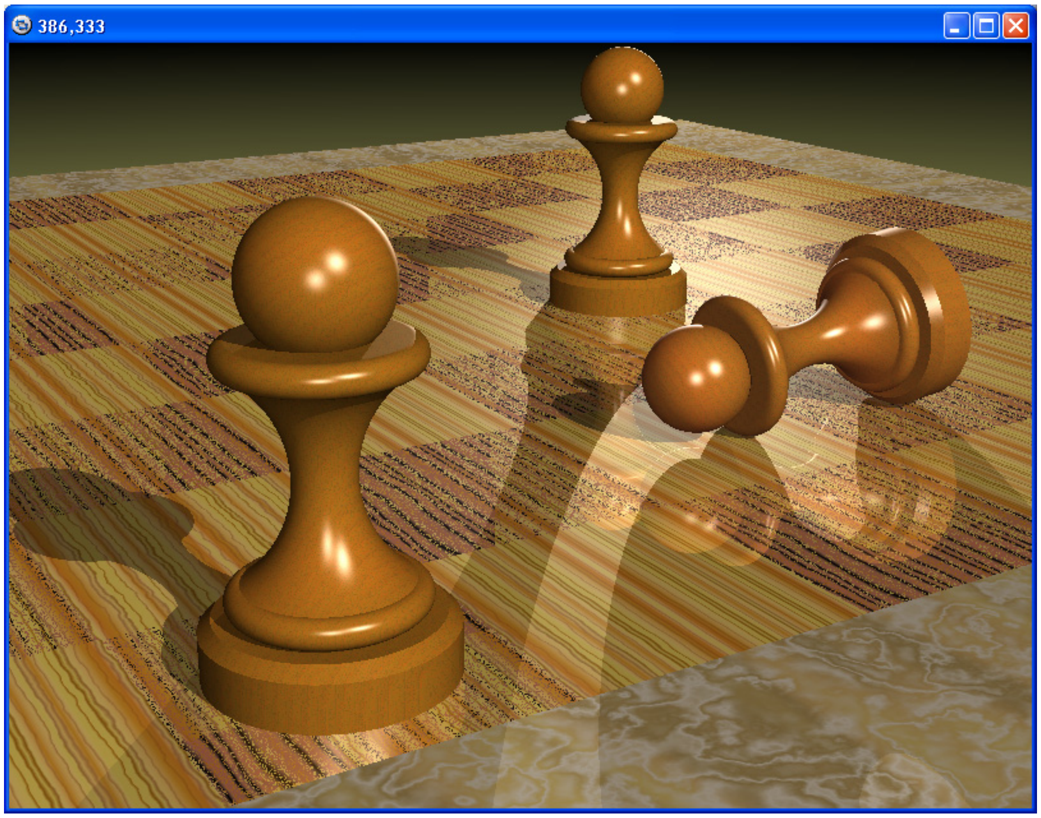


Figure 34:

A closeup on one of the pawn. (Figure 35)

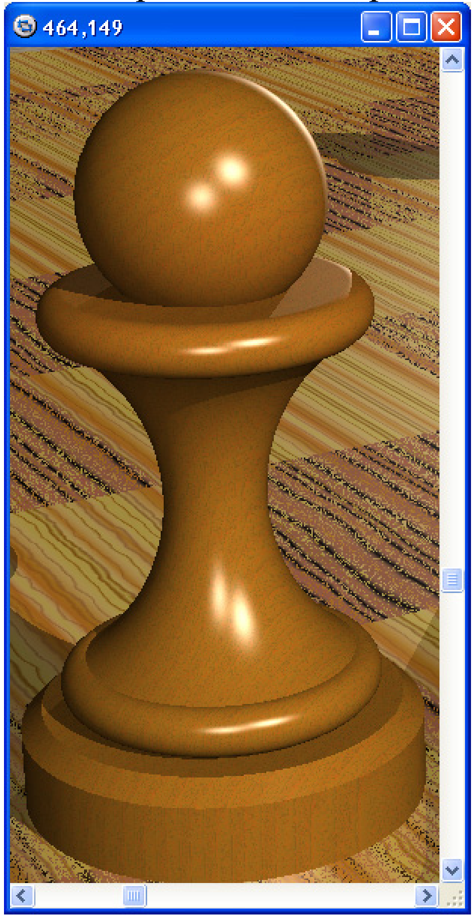


Figure 35:

Below is a scene with a Kauri-made desk. (Figure 36)

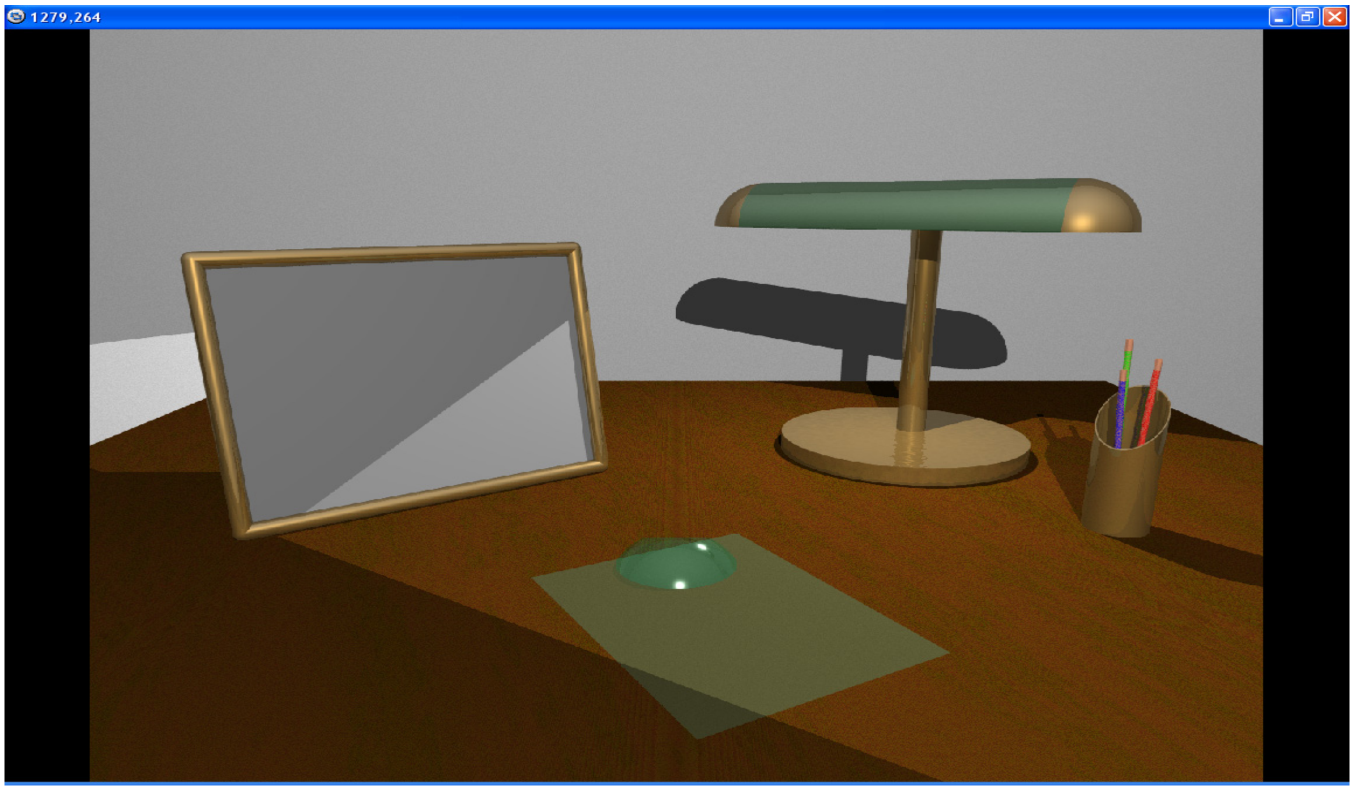


Figure 36:

5.2.2. Rimu

Here are the same scenes used for Kauri, but with Rimu applied, and comparisons can be made. (Figure 37)

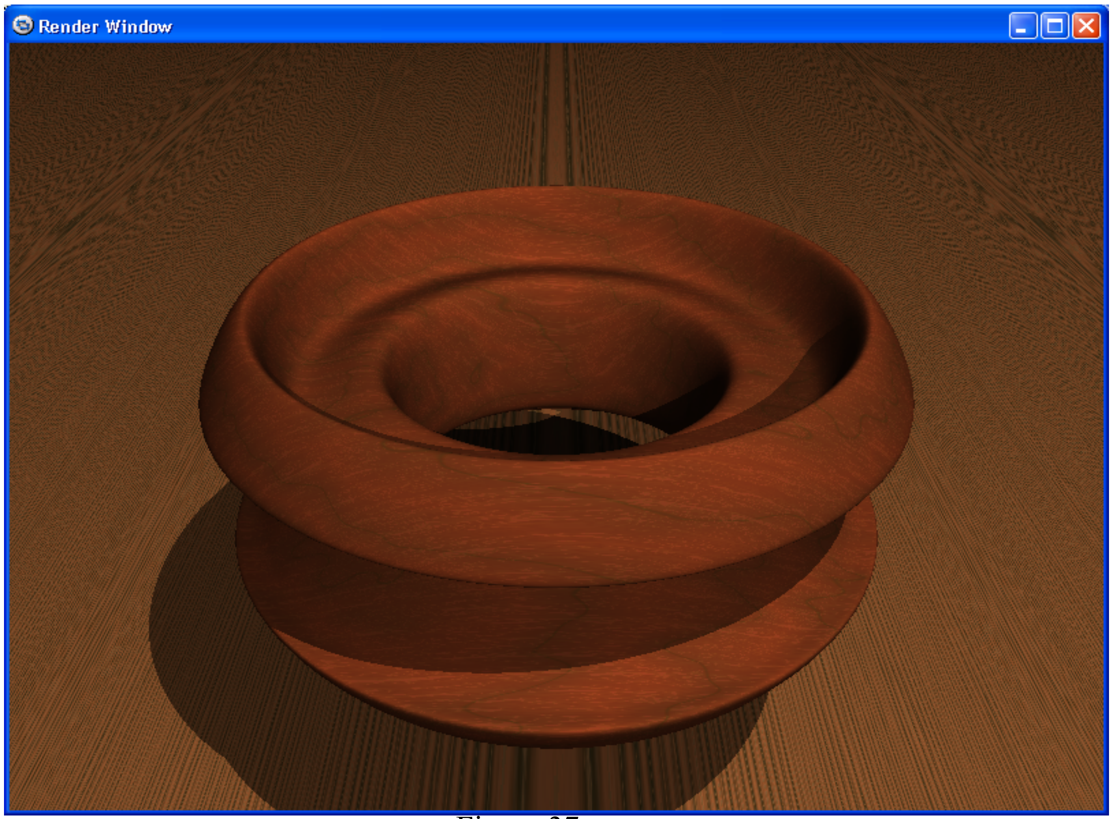


Figure 37:

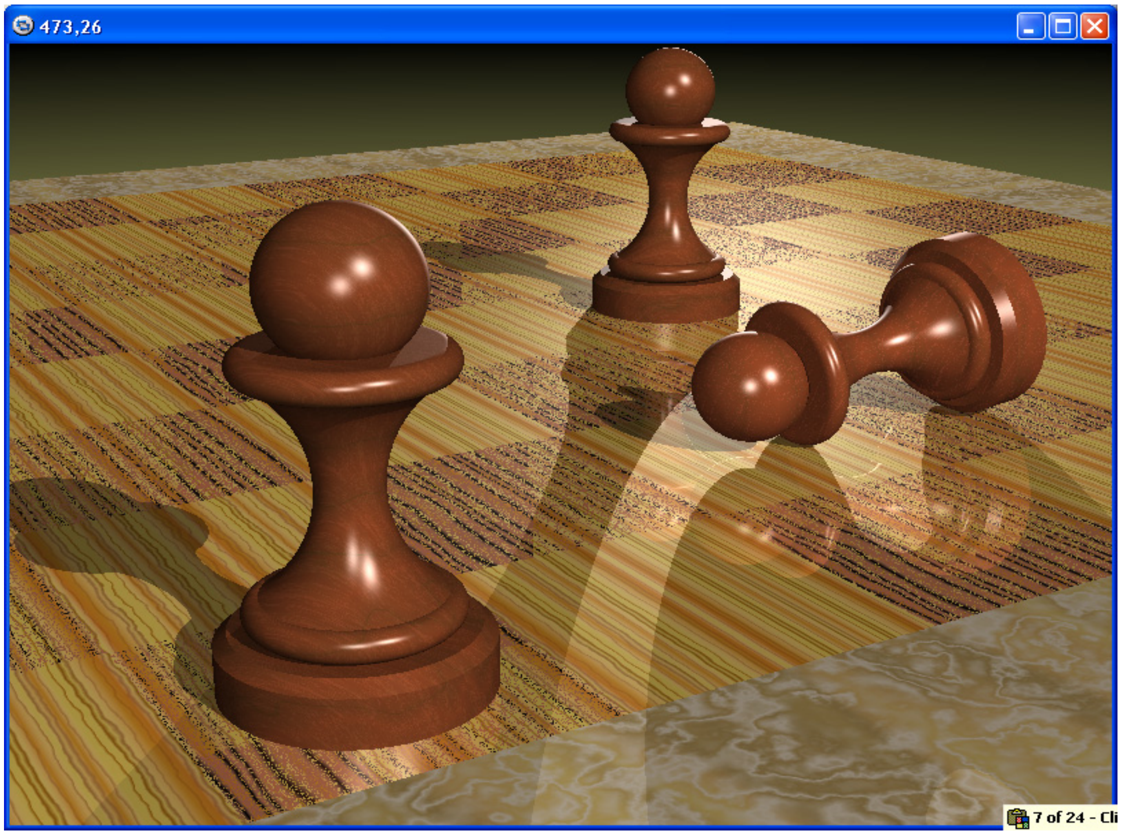


Figure 38:

6.Conclusion

6.1. Most suitable tool/software used

I finally end up developing wood texture in POV-Ray, which was unexpected. In POV-Ray, there are already many useful functions like Turbulence implemented, which made the process easier, but in order to generate realistic wood texture, we need to study the characteristics of the wood in detail; to achieve the best result, slice the wood in different direction and examine the colour, the way the grains go, and any additional features of the wood, all these factor need to be carefully examined and considered.

6.2. Steps in modeling wood texture

Here are some steps I find essential for generating a quality wood texture, I will list them out.

1. Colour sampling; use a scanner to scan the wood surface from many angles. From the images, find the colour range and hue range for later, very useful for choosing colours.
2. Define layers; examine the wood and figure out how the texture is composed, divide them into different layers. E.g. Year ring layer, pore layer, or a sheen layer.
3. Apply to objects and compare with the real objects.
4. Adjust if patter and color little by little to achieve best results.

6.3. The layering and the application of algorithms

From my practice of texture modeled, I realized that it is much of an art, and just like realism painting, it requires good eyes of colour selection and fine observation of details. Despite the common artist skills for painting, the two most important and the out-of-ordinary techniques in which is the layering and the application of algorithms. Firstly in layering, we must realize the different elements there are in the wood, and some of the characteristics combined can not be manipulated by the algorithms to model them precisely. Eg. The light and blurry year rings and the dark but sharp pores of the wood. They must be modeled separately, and then combined later. Secondly, is to apply suitable algorithm with the right value of parameters. This is the most difficult part out of all, but I have to say that the trick is to understand what exactly each of the algorithms does and what effects they produce when applying different value of parameters. In POV-Ray, the examples are turbulence, omega, and octaves. From the textures I have modeled in POV-Ray, and in turbulence in particular for example, I find no correlations between the values the parameter and the measurements taken from the scanned wood images. It was pretty much trial and error and estimated by eyes. And essentially, it's all comes down to experience and practice.

6.4. Ideas Learned

From this research, we learn that turbulence is more suitable to generate wood texture rather then having just noise alone, mainly because turbulence has the effect of controlled-distortion, very suitable to model grains, as noise alone is more suitable to generate textures like marble to suit its smooth-randomness.

And surprisingly, we find that the hue value of the Kauri varies only by 11, which makes colour selection much easier, and most likely same apply to other woods, due to find out.

The two native woods, Kauri and Rimu, they have a common feature, their grains are less obvious compare to the heavily grained Radiata Pine, which means the following, they grow steadily throughout winter and summer, mainly because they are very tolerable to temperature and moisture change, I will imagine since it's NATIVE, surely it adapts to its environment well and the imported Radiata Pine has much more obvious grains.

6.5. Improving quality

Knots can be added to improve realism.

7.Future Research

Possible future related research:

1. Virtual wood carving tool.
   - Build on existing 3D texture, and develop a realistic wood carving software.
2. Generating realistic wood knots in fit into textures.
   - Knots can improve the realism of a texture if done properly.

8.Acknowledgements

I want to thank my college woodwork teacher Mr. Bordot for providing facility for wood processing, to help me to get more

accurate and better results from scanning the wood block, without the help of the buzzer (thicknesser), the job would be a lot harder.

9.Bibliography

[1] Perlin Noise
http://freespace.virgin.net/hugo.elias/models/m_perlin.htm
[2] Texture Wrapping
http://www.cs.auckland.ac.nz/compsci372fc/lectures/372NotesHandout6_1up.pdf
[3] How to use wood textures in POV-Ray
http://home4.inet.tele.dk/ibras/povtips/povwood.htm
[4] Noise, Turbulence, and Texture
http://cs.wpi.edu/~matt/courses/cs563/talks/noise/noise.html
[5] Texture Mapping
http://www.sgi.com/grafica/texmap/
[6] Kauri
http://www.learnz.org.nz/2001/great_barrier/kauri.htm