**INDIAN INSTITUTE OF TECHNOLOGY DELHI**

DEPTT./CENTRE: **COMPUTER SCIENCE AND ENGINEERING**

REPORT OF THE PROJECT SHORTLISTED UNDER **SURA - 2015**

Project Titled: **Metal Corrosion Rendering**

**Submitted by:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name of Student** | **Department** | **Entry No.** | **Email (Contact no.)** | **Signature** |
| Rishit Sanmukhani | CSE | 2013CS10255 | cs1130255@iitd.ac.in (9711932719) |  |
| Yash Gupta | CSE | 2013CS10302 | cs1130302@iitd.ac.in (8588824291) |  |

.

Name and signatures of the Facilitators: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Subodh Kumar

Department: Computer Science Engineering

Mobile no. of the Facilitator: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date of submission of report in IRD:

Table of Contents

[Acknowledgements 2](#_Toc433923335)

[Chapter 1 3](#_Toc433923336)

[Introduction 3](#_Toc433923337)

[Chapter 2 4](#_Toc433923338)

[Methodology 4](#_Toc433923339)

[Identification of parameters 4](#_Toc433923340)

[Extraction of Parameters 5](#_Toc433923341)

[Algorithm 6](#_Toc433923342)

[Generating texture maps 8](#_Toc433923343)

[Algorithm 8](#_Toc433923344)

[Rendering 9](#_Toc433923345)

[Chapter 3 10](#_Toc433923346)

[Experiments & Results 10](#_Toc433923347)

[Experiments with Color Space 10](#_Toc433923348)

[Experiments with MAX\_NEIGHBOURS 11](#_Toc433923349)

[Experiments with Control Points 14](#_Toc433923350)

[References 16](#_Toc433923351)

# 

# Acknowledgements

We would like to thank Prof. Subodh Kumar for advising us enthusiastically throughout

the course of the Summer Undergraduate Research Project (SURA). We would also like to thank him for giving us this opportunity to work on this exciting project, and for encouraging us at all times. We would also like to thank Nisha Jain (Research scholar) for providing key insights throughout the course of the project.

Rishit Sanmukhani (2013CS10255)

Yash Gupta (2013CS10302)

Indian Institute of Technology Delhi

# Chapter 1

# Introduction

One of the key areas in the field of computer graphics is realistic rendering of corroded objects. There are several approaches including physical modelling, extracting characteristics from actual photographsThe main challenges are to identify important features underlying beneath the image, extracting those features and creating object models in accordance to it.

Our objective in this project is to produce realistic renders of metallic object in corroded state. We are given mesh representation of the object in corroded states

# Chapter 2

# Methodology

## Identification of parameters

First step in realistic rendering pipeline was to identify the parameters representing appearance of object model. Parameters like diffuse color, metallicity, smoothness, bumpiness, depth information, reflectance play a major role in rendering of object model. For example, diffuse color will determine the final color of rendered object model, bumpiness will determine irregularities at the surface level and depth information will give displacement in the polygon mesh representation of the object model. This makes parameter selection an important step.

We decided to use Unity3d -5 for rendering the object model. Motivation for using Unity3d-5 was because it has many built-in shaders and parameters of those built-in shaders can be changed. This saves a lot of writing custom shader, as we can use Unity3d built-in shader as base for our rendering purposes. Unity3d also provides option of tweaking the built-in shader by a script (which can be written in JavaScript or C#).

Following two built-in shader offered us best set of parameters for our object model:

1. Standard Shader with metallic setup
2. Standard Shader with Specular setup

Both of above two shaders were major addition in Unity3d-5 edition replacing large number of old built in shaders. Our main objective was to render metallic object in corroded state, so we decided to use Standard Shader with metallic setup. It gave us right set of parameters for our object model.

We identified and worked on following parameters offered by standard shader with metallic setup:

1. Albedo

Albedo parameter offers you control over base color of the surface. We had an option of providing single color or providing a texture map for our Albedo parameter. Later option gives us more control over appearance of surface of final rendered object model.

We call the texture map assigned to Albedo parameter as Albedo Map.

1. Normal Map

This is a type of offers you control over surface details such as bumps and grooves. We have to provide a texture map (of type normal map) representing fine surface details of the object model. It is important to mention here that normal map does not alter the polygon mesh geometry of the object model, rather it just modifies the normals as per the texture map values. This modification of normals determines how lightning is calculated on the surface and thus giving appearance of embossed surface details.

We use Unity3d to convert our texture map to normal map. We call our texture map as Depth Map.

1. Metallic

Metallic workflow are modified by following two parameters:

1. Metallicity

This parameter defines how “metal-like” the surface is. More is the metallicity, more are the reflections from the surrounding making albedo color less visible.

1. Smoothness

This parameter defines the smoothness of the surface. Smoother the surface, lesser are the microsurface details and such a uniform surface reflects light in narrowly focused area. Because of it reflections of object environment are clearly visible. But if smoothness level is very low, then light is reflected in very wide area, thus averaging out and reducing reflections.

Thus both of the above parameters alter the reflectivity and light response of the surface thereby modifying metallic nature of object model. We have option of assigning the same value of metallicity and smoothness to entire object model by slider or using a texture map. We prefer using texture map as it gives helps in capturing the variation of metallic nature of object model. In the texture map we use, Red channel is used as metallicity and alpha channel is used as smoothness. (This is as per Unity3d convention)

We will call this texture map as Metallic Map.

There are number of other parameters which we could have worked upon, but in this project we considered using above four parameters. This gives us sufficient variation in capturing the appearance of the object model.

Due to corrosion, some part of the metallic object gets removed and thus geometry of the object changes. If we consider polygon mesh representation of the object model, depending upon corrosion degree, voxel of this mesh gets removed due to corrosion. To capture this, we need to change to geometry of the object model depending on the corrosion degree and none of the above mentioned parameters alter the geometry of the object. So we generate a texture map which gives us amount by which geometry of the object will get altered. We call this texture map as Displacement Map.

Following is the pseudo code of script which uses Displacement Map and modifies mesh geometry:

Texture2D tex = load Texture

Mesh mesh = load Mesh

float factor = -0.5f

for (int i=0 ; i<mesh.vertices.length ; i++)

mesh.vertices[i] += mesh.normals[i] \* tex.value \* factor;

mesh.RecalculateNormals();

## Extraction of Parameters

We have identified following parameters to represent appearance of object model:

1. Albedo Map
2. Depth Map
3. Metallic Map
4. Displacement Map

We introduce a new term “weathering map” which represent the corrosion degree of the input object model. Each pixel of weathering map will have a value between 0 and 1 (0 has least corrosion degree). Corresponding to the weathering map, we need to generate above four maps to render object model in Unity3d. But in order to generate above four maps, we need to have information about the values of different parameters at different corrosion degree before hand. In this project we will extract information but restricting ourselves about the parameter “Albedo value” from sample input image of corroded metallic object.

### Algorithm

We present following technique for extraction of Albedo(Color) values:

1. User interaction

Our methods requires a set of points(pixel coordinates) from the image with their corrosion degree. These set of points are called control points. Control points gives us the idea about color of the object at particular corrosion degree. These control points are taken as input from the user. Number of control points are not fixed and user can provide as many as he wish. In general, higher the number of control points, better will be correlation between the color and corrosion degree.

1. Generating graph

We create histogram of the given image to evaluate unique colors present. We sample down the color to reduce the computation in the subsequent steps. Unique color calculated in this step form the nodes of our graph.

In order to decide the neighbours of a node in the graph, we calculate distance of each node to given node and if it is within a set threshold (NEIGHBOUR\_THRESHOLD), we add an edge between the nodes with weight as the distance we just calculated. This distance is the euclidean distance between colors in RGB space or YUV space (configurable). For each node in graph, we take the nearest MAX\_NEIGHBOUR which fall within NEIGHBOUR\_THRESHOLD.

1. Generating corrosion-albedo mapping

We find the nodes corresponding to the given control points, and assign the corrosion degree given by user to these nodes. These values are the basis for assigning corrosion degrees to other nodes.

Now, considering each control point as source node, we calculate single source shortest path distance to all other nodes using Dijkstra’s algorithm. For each node, we have the shortest-path distance to each control-point node in the graph. We then calculate a weighted sum of corrosion degrees of the control-point nodes based on the distance to respective control-point node. We use weights inversely proportional to the distance. This gives an effect of corrosion degree leaking around the control-point node, with each control-point node maintaining its value as given by the user.

There may be some control-point nodes which are at infinite distance from a given node, which means that the control-point node and the given node lie in different disconnected components of the graph. In this case, the corrosion-degree of the given node depends only on the control-points which lie in it's connected component in the graph. This results in better results as we do not try to explicitly relate control-points which do not have a gradual gradient to each-other in the colors of the image. The level of connectedness is decided by the NEIGHBOUR\_THRESHOLD along with distance metric we used for colors.

After the last step, we have corrosion-degree values for each node in the graph. Now we calculate inverse mapping from corrosion-degree(down-sampled to the number of levels we want in output gradient) to color value of the node. This results in a one-to-many mapping, with multiple colors for each corrosion-degree. In our implementation, we simply square root the average of square of color values of each node that maps to a single corrosion value and obtain a one-to-one mapping between corrosion-degree and color. However, we still have corrosion degrees for which no color-nodes existed. This results in undefined colors for several corrosion-degrees.

To assign color values to these corrosion-degrees, we use linear interpolation between the nearest known corrosion-degree - color pairs. This completes our one-to-one corrosion-albedo map, which we finally output as a gradient where color at top corresponds to 0% corrosion degree and color at bottom to 100% corrosion degree.

1. Feedback

Using the corrosion-albedo map generated in above step, we generate weathering map corresponding to the input image. For each pixel of input image, we calculate the nearest color from the map. If this nearest color is within NEIGHBOUR\_THRESHOLD, we assign the pixel corresponding corrosion degree. Else we just mark the pixel red, which depicts that given color was not captured by the control points marked by the user. User can accordingly modify the control points.

## Generating texture maps

We now generate following maps to produce realistic render of object model:

1. Albedo Map
2. Depth Map
3. Metallic Map
4. Displacement Map

To generate above maps, we require mapping of corrosion degree to corresponding parameters represented in the map. Using the above mentioned technique we generated corrosion-albedo mapping from some sample input image of corroded metallic object. The remaining maps were empirically designed according to rendered results (to limit scope of the project).

### Algorithm

We present following technique for generating above maps corresponding to a given input weathering map:

We take a weathering map as input and multiply each pixel's color with a given noise map (with average value 0.5) such that each corrosion-degree changes by a maximum set percentage ('alpha').

*degree' = ((1-alpha)+noise\*alpha\*2)\*degree*

Apart from the 'base' and 'noise', we also take appearance manifold in the form of corrosion-albedo map, corrosion-depth map, corrosion-displacement map, corrosion-metallicity map and corrosion-smoothness map. Then a simple mapping is done from *degree'* to albedo, depth, displacement, metallicity and smoothness based on input maps to generate corresponding texture maps.

## Rendering

Using the above mentioned technique, we know have texture maps corresponding the parameters which represent appearance of the object. We know take polygon mesh representation of our given input object model, and render final results in Unity3d.

As mentioned previously, we use standard shader in the unity with renderer mode of shader set to opaque. We also assign our generated texture maps to corresponding parameters of standard shader in unity3d.

# Chapter 3

# Experiments & Results

## Experiments with Color Space

In our algorithm, we experimented with distance metric (used to create edges in graph) and color space models. The motivation behind different color space model (RGB / YUV / LAB) was to create a graph in which dissimilarity among different nodes (colors) is in correlation with visual dissimilarity. We used naive euclidean distance metric and CIE76 distance metric, with later producing more convincing results.

We found best results using LAB color model with CIE76 distance metric as it mimic the nonlinear response of eye.

|  |
| --- |
| D:\Downloads\SURA-2015-10-29\SURA\RGB.bmp  RGB Space Distance |
| D:\Downloads\SURA-2015-10-29\SURA\YUV.bmp  YUV Space Distance |
| D:\Downloads\SURA-2015-10-29\SURA\LAB.bmp  L\*ab Space Distance |

## Experiments with MAX\_NEIGHBOURS

MAX\_NEIGHBOURS is a threshold which is used to limit the degree of each node (color) in the graph. On experimenting, we didn’t find much difference if value is set above 10. But if the value is too low(4-5), the results were not convincing. The main reason is that for very low value, each color doesn’t have sufficient neighbours to capture different colors resulting in sparse graph. When this value increases, number of different colors among neighbourhood increases making graph more useful for computation of corrosion values.

We found the value of 12 as the optimal one, as it makes graph sufficiently connected and also doesn’t make computation too expensive.

|  |
| --- |
| RGB Space Distance |
| YUV Space Distance |
| L\*ab Space Distance |

## Experiments with Control Points

In our algorithm, we took control points as input from the user. As mentioned previously, higher are the control points better are the results because it gives us good initial idea about relation between corrosion degree and color. So incase user makes mistakes in entering control points we generate feedback which gives idea about which color are captured.

|  |
| --- |
| D:\Downloads\SURA-2015-10-29\SURA\RGB.bmp  RGB Space Distance |
| D:\Downloads\SURA-2015-10-29\SURA\YUV.bmp  YUV Space Distance |
| D:\Downloads\SURA-2015-10-29\SURA\LAB.bmp  L\*ab Space Distance |

## Experiments with Perlin Noise

## 

# References

1. <http://doi.acm.org/10.1145/1179352.1141951>
2. <http://web.eecs.utk.edu/~huangj/papers/polygon.pdf>
3. <http://etd.uwaterloo.ca/etd/rjaroszkiewicz2003.pdf>