VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

DECLARATION

I hereby declare that the project entitled **"Into the Virtual World with WebGL"** submitted by me to the School of Computer Science and Engineering, Vellore Institute of Technology, Vellore-14 towards the partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering** is a record of bonafide work carried out by me under the supervision of **Prof. AJU D., Associate Professor, School of Computer Engineering**. I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or university.

Signature

Name: AKSHANSH MANCHANDA

Reg.No: 14BCE0579

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

CERTIFICATE

The project report entitled "**Into the Virtual World with WebGL**" prepared and submitted by **AKSHANSH MANCHANDA (14BCE0579),** has been found satisfactory in terms of scope, quality and presentation as partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering** in Vellore Institute of Technology, Vellore-14, India.

Guide
AJU D.

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

I am thankful to Prof. AJU D. to be there whenever I needed another opinion on ‘How to get the work done?’ and ‘Where to look for ideas and resources?’. His experience in the same field was handy whenever I was in need of clarity on certain aspects of the project. I would also like to thank the HOD and Dean of the SCOPE Department to oversee the administrative tasks involved in documenting, consolidating and regulating the large amount of projects being worked on by many students of VIT University. I express my gratitude to VIT University which has given me a platform to pursue a project that is an important step in the journey of realizing my ambitions. I owe the knowledge I have today to the book ‘Real-Time Rendering’ and the Udacity Course ‘CS291 – Interactive 3D Rendering’. Lastly, I couldn’t have done the project without the loads of examples available on the GitHub repositories and tons of textures freely available, courtesy of many artists with experience in capturing and generating such textures.

TABLE OF CONTENTS

S.No.	**Title**	**Page**
I	Title Page	i
II	Declaration	ii
III	Certificate	iii
IV	Acknowledgement	iv
V	Table of Contents	v
VI	Abstract	vi
1.	Introduction	1.
2.	Literature Review	24.
3.	Overview of Proposed System	27.
4.	Proposed System Analysis and Design	29.
5.	Results and Discussion	32.
6.	Conclusion, Limitations and Scope for Future Work	32.
VII	List of Figures	vii
VIII	List of Abbreviations	ix
IX	References	x

ABSTRACT

The field of graphics programming has seen huge development in the last 15-20 years. Many applications like video games, pre-visualizing movie scenes, creating interactive web pages have been influenced by and in turn have influenced the area of research called real-time rendering. While several algorithms strive to achieve high performance and quality, applications are limited by performance capabilities of the device used on the consumer end. Hence, graphics programming is not limited to mimicking reality with absolute precision but expanded to various techniques that simplify actual physical phenomenon and use functions at various stages of the rendering pipeline that aim to create an interactive and believable experience. Using effects like transparency and shadows have their own limits and limitations, which requires optimization at various levels. WebGL, a graphics API based on OpenGL ES, is capable of performing In-Browser GPU-based rendering and hence is bound to open a new paradigm of applications as well as attract new developers to this paradigm. A JavaScript Library called THREE.js has made this field more accessible to new developers and easier to experiment with for existing developers. It has easily available resources for widely used graphics techniques and is frequently modified by interested parties in the GitHub environment. It is still a work in progress, but is still an excellent place to work on a browser-based product or just learn. This project intends to create a demo scene to be a decent demonstration of the kind of capability it has.

1. Introduction

1.1. Theoretical Background

- Coordinate System, Vectors and Matrices:

- Points and vectors have x, y, z components and a w component which is 1 for points and 0 for vectors.

- The distinction in points and vectors is that points are used for specifying positions in the world space whereas vectors refer to a movement in world space.

- We only add vectors in practice as combining movements makes sense but combining positions does not. Even if we add two points, we will get w = 2, which represents neither a vector nor a point.

- THREE.js supports 2, 3, and 4 dimensional vectors as Vector2 ( ), Vector3 ( ), Vector4 ( ). A Vector4 with w = 1 represents a point usually.

- Matrices for Translation, Rotation and Scaling Transforms.

$$T = \begin{bmatrix} 1, & 0, & 0, & t_x \\ 0, & 1, & 0, & t_y \\ 0, & 0, & 1, & t_z \\ 0, & 0, & 0, & 1 \end{bmatrix} \qquad R_z = \begin{bmatrix} \cos(a), & -\sin(a), & 0, & 0 \\ \sin(a), & \cos(a), & 0, & 0 \\ 0, & 0, & 1, & 0 \\ 0, & 0, & 0, & 1 \end{bmatrix} \qquad S = \begin{bmatrix} S_x, & 0, & 0, & 0 \\ 0, & S_y, & 0, & 0 \\ 0, & 0, & S_z, & 0 \\ 0, & 0, & 0, & 1 \end{bmatrix}$$

- THREE.js has functions for these operations, always executed in the order 'Scale then Rotate then Translate', useful for most situations where we want to first rotate about the origin then move. For other order of transforms we copy the mesh to a new object using the method Object3D ( ).

- THREE.js provides a World View(W) and Model View(M) Matrix for each object which can be used for transforming by performing matrix operations.

- Colour Space:

- All calculations, we deal with, are generally done in Linear RGB Colour Space which is an additive colour space i.e. mixing colours gives brighter colours.

- The RGB colour space is limited in the sense that it can’t display some pure spectral colours and such colours are depicted by most appropriate mix of Red, Green and Blue.

- The calculations done by us are in linear RGB space, which does not match the colour space displayed by the monitor and the colour space perceived by us as these two non-linearly correlate to the linear RGB colour.

- We need to account for this in our calculations to avoid perceptually wrong looking colours and unnecessarily aliased graphics.

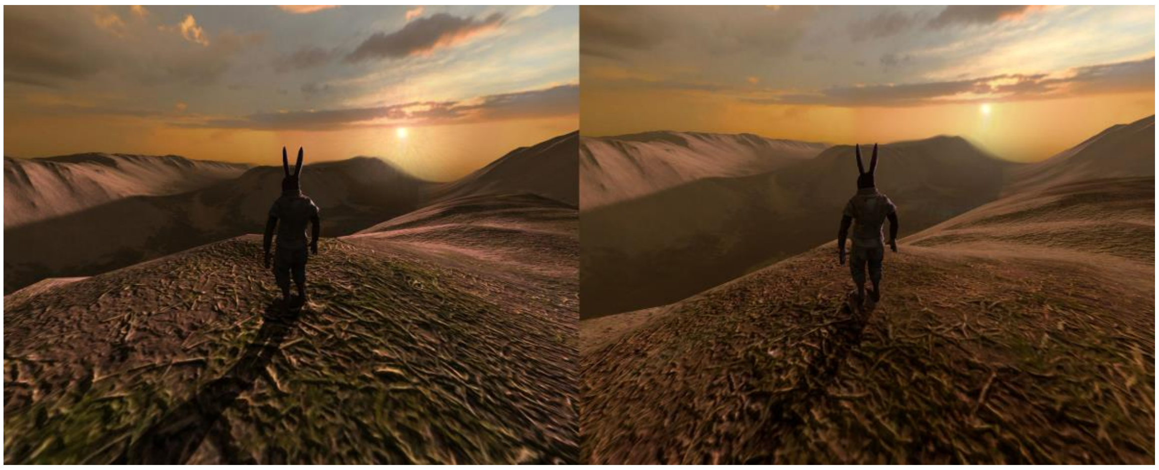

Fig. 1: Comparison: No Gamma Correction vs. Gamma Correction

- The range of colours for a display or a printer is called the colour gamut of that device. Below is an image showing gamut of some devices.

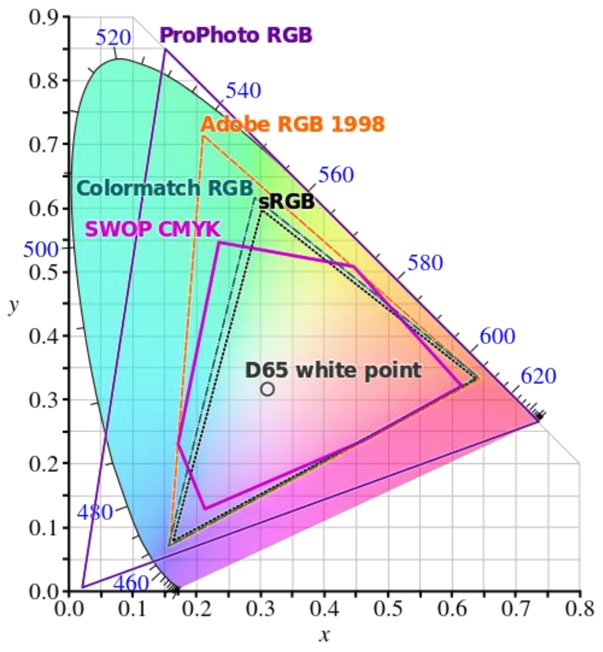

Fig. 2: Different Colour Spaces

- It is worth noting that the above picture is repeating colours outside a certain range, as the device it’s being viewed on will have a limited colour gamut. The image also would have been made in a limited colour space to support many devices.

- Gamma Correction solves the problem of linear and non-linear spaces not being linearly correlated.

- This is done by raising the original colour to a power of 2.2 to get the linear RGB space colour. After the operations, we get the correct colour back by raising the colour to the power 1/(2.2)

- This leads to another problem as instead of the 255 levels of colour, we had, we get 183 levels of colour leading to less convincing shading.

-This is generally solved by using more bits to represent colours. One way this is commonly done is using HDR lighting and textures.

- THREE.js supports Boolean values gammaInput and gammaOutput and by default uses 2 instead of 2.2 for conversion.

- Meshes and Models:

  - A Geometry is a set of vertices and faces made using those vertices.

  - Triangular faces is the language that most of the current GPUs generally use. The advantage of a triangle is, it is by default in a plane (unlike other polygons) and so calculations are easy.

  - A Material contains a variety of information that affects the look of any object it is applied to.

  - A Mesh is a combination of a Geometry and a Material.

  - Complicated objects can be tedious to make by programming, so a Third-Party Modelling Application is used and then the model is exported to be used.

  - Procedural Modelling is another concept where objects are created by programming.

  - THREE.js provides geometry with attributes like vertices, faces, and uvs in array form.

  - THREE.js provides some basic templates for materials with adjustable parameters.

  - THREE.js provides a Mesh constructor which takes geometry and material as input.

- <u>Camera</u>:

- A camera is defined by the camera position and the direction of the camera or target coordinates.

- An orthographic camera is a camera that shows the objects image as the absolute function of its size, in other words it ignores the effect in which objects appear smaller, the farther they are from the camera.

- Such a camera is commonly used in architecture plans to show measurements and in 3D modelling software for ease of editing.

- A perspective camera shows the size of an object decreasing with depth and hence looks like how we see things in reality. An image formed by such a camera is more believable and hence most often used at consumer end.

- An example showing three orthographic views (top, front, left) and one perspective view, being used in 3DS Max 2018, is given in the picture below.

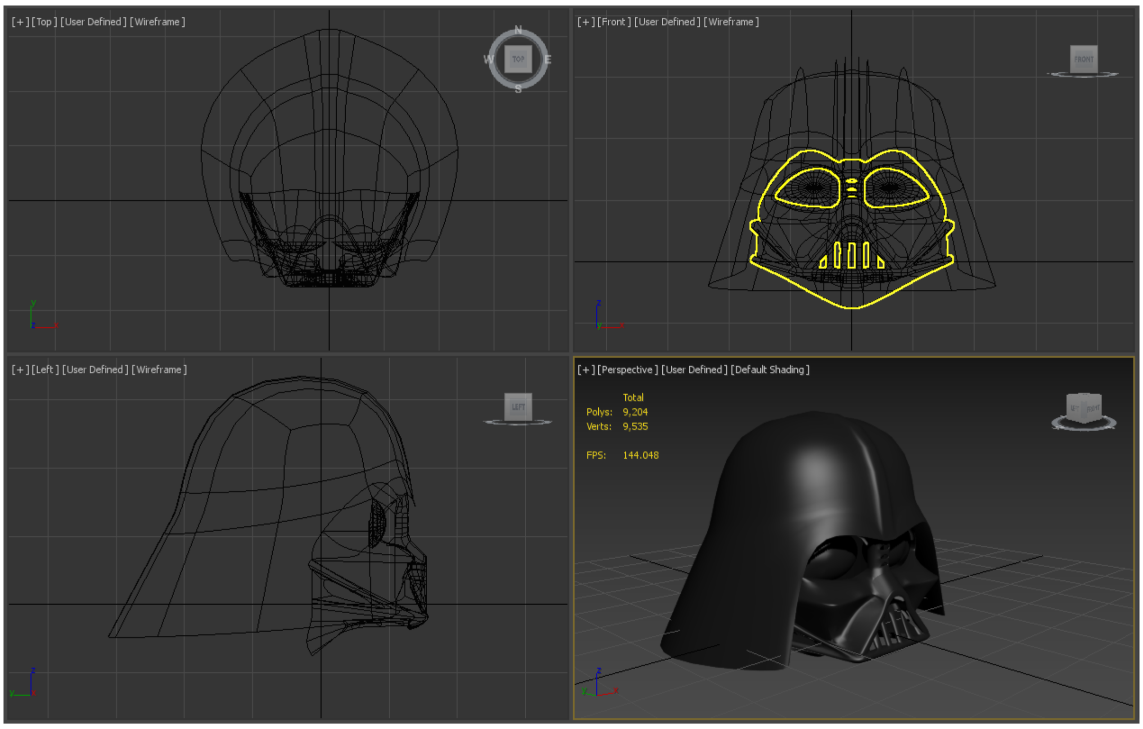
[+] [Top] [User Defined] [Wireframe]
[+] [Front] [User Defined] [Wireframe]
[+] [Left] [User Defined] [Wireframe]
[+] [Perspective] [User Defined] [Default Shading]
Total
Polys: 9,204
Verts: 9,535
FPS: 144.048

Fig. 3: Darth Vader Mask in 3DS Max

- An orthographic camera has a top, bottom, left, right, near and far plane to define what will appear in the scene and what will not, forming a Bounding Box.

- The perspective camera defines the visible volume by using near plane, far plane, field of view and aspect ratio.

- A greater field of view, means more of the world is visible as the visible volume diverges more with distance.

-This visible volume is also called a view frustum and looks more like a square pyramid as opposed to the bounding box.

- Materials and Lighting Contributions:

- Materials define how a material looks by storing information about how the rough the surface is, how it reacts to light etc.

- In real world, light comes via a light source, bounces around and reaches our eye. Many rays are involved and computing this model is expensive.

- So the model we use simplifies the real model. All calculations are done as if rays are being shot from the camera and on hitting a surface, a colour is generated.

- Usually light primitives are used for different type of light like directional light, point light, ambient light, spot light, area light etc. To add realism to lights, we need to understand the flow of light energy that comes under the discipline of Radiometry.

- Another area of study is Colorimetry which deals with this energy in terms of colours. This adds more realism, as it accounts for the sensitivity of our perception to different colours and it has inter-convertible terms for all radiometry concepts.

- Directional light is a light source that acts as if it is infinitely far away i.e. Light rays are parallel and hence is a good way to show sun light. Point lights are good to represent how light energy drops with distance and hence is used to represent light sources like bulbs.

- The simplified model we use for lighting in real-time, disregards the effect of light after it bounces from one object. Mathematically, the final colour is computed by adding 4 types of lighting contributions.

- Final Colour =
Emissive + Ambient + $\sum$ Diffuse (Light) + $\sum$ Specular (Light, View)

- Emissive is a way to paint a light source with a constant solid colour and it ignores the effect of other objects on the light.

- Ambient is a way to add a constant colour to the whole scene, this is done to add some colour to areas which will otherwise be black as our model ignores bouncing of light.

- These two components do not depend on lights in the scene or view direction.

- Diffuse relates to the colour an object doesn’t absorb and is computed by adding contributions of all the lights in the scene.

- Specular is used to add shininess to an object. Each light causes a shine and the shine moves with the view direction and so specular is a function of lights in the scene and the view direction.

- All types of simplifications have a physical analogy. We deal with two kinds of effects related to light, one is the effect of an object’s surface and one is the effect of the particles embedded below the surface.

- When we talk about the surface we deal with light being reflected by small irregularities in the surface that are too small to see called micro-facets.

- When we talk about the particles beneath the surface we deal with light bouncing by particles and finding its way back out. This is called subsurface scattering.

- Diffuse contributions try to account for light that is transmitted, absorbed and scattered. Physically, what happens is that the subsurface particles scatter light in many directions and hence we simplify this as a colour vector visible for all directions.

- Micro-facets of smoother surfaces reflect a lot of the light giving that shine and hence the specular contributions. We also specify a specular colour which is usually white, with exceptions like copper.

- Micro-facets for rougher surfaces cause bouncing of light by many micro-facets resulting in loss of energy. This results in blurrier specular highlights with lower intensity but larger area.

- What happens beneath the surface in many materials is that the particles are so tightly packed that the position at which ray enters the object and comes out is almost the same.

- So we assume these two positions to be exactly the same and colour calculated at a pixel can be used directly for that pixel. This assumption works reasonably well in practice but there are cases where we need to do additional computations like light escaping wax after subsurface scattering.

- More realism is obtained by accounting for local subsurface scattering for insulators using functions on the constant albedo used to represent scattering of different materials. For metals, we need to account for the micro-geometry, which is light reflecting from mirror-like micro-facets, using BRDF functions.

- THREE.js provide different materials which show different balance of performance and quality. These are good for a large number of materials so we wouldn’t dig too deep for this project, but an understanding of the physical concepts is essential if one wants to improve performance and quality by using programmable shaders(discussed later).

- Recently, physically correct shading has gained popularity as sometimes it is better to do these calculations than use several effects on non-realistic shading to get similar results. THREE.js has some materials that have similar properties as materials in Unreal Engine (a widely used gaming engine) which look amazing in many situations.

- For light contributions, the half-angle vector is calculated, which for a surface is the vector which divides the angle between incident and reflected rays.

- The light contributions specular and diffuse increase with the cosine of the angle between the shading normal (discussed under next topic) and half-angle vector.

- It is worth noting that phenomenon happening here is happening at a scale a lot smaller than a pixel and it can’t be seen in reality too, but it does change the look of a material in ways that need to be accounted for.

- Shading and Illumination Model:

- Models usually are a large number of triangles. This is called triangulation and extended to other geometric primitives, tessellation. Using the geometric normal of such faces, for shading calculations, will result in flat looking faces.

- To solve this, the concept of shading normal is used, which are usually stored at vertices and interpolated over the surface by the built-in parts of the graphics pipeline to give a smooth looking surface.

- Two basics types of shading that have been used for a long time are Gourad Shading and Phong Shading.

- Gourad Shading stores colours at vertices and the GPU interpolates these values for each face using the vertices. Phong Shading interpolates these colours per-pixel, which gives a more accurate look, eliminating artefacts like sudden change in shading at a performance cost.

- Another part of the calculations done is the Illumination model. Lambertian model evaluates light contributions for non-shiny surfaces (no specular) per-vertex. Blinn-Phong model evaluates these contributions per-pixel at the cost of performance and also supports shiny surfaces.

- Lambertian Diffuse Term (in correspondence to Lambert’s Law which states outgoing radiance is proportional to cosine of angle between shading/surface normal and light vector, for an ideal diffuse material) :

$$L_{diff} = (c_{diff} / \pi) \ *' \ (E_L * \max(\cos\theta_{nl} , 0))$$

- Where *’ is piecewise vector multiplication operator, denoted by a cross inscribed in a circle, that generates vector components by multiplying x with x, y with y and z with z to give x, y ,z components respectively. Except π all other are radiometry constants decided by the light or material.

- Now if we add the specular term that needs to account for viewer’s direction, we get the radiance term according to Blinn-Phong Model. This final result is as follows:

$$L = L_{diff} + L_{spec} =$$
$$(\ (c_{diff} / \pi) + ((\max^m(\cos\theta_h , 0))*((m + 8) *(c_{spec})/(8 * \pi))) \ ) *'$$
$$(E_L * \max(\cos\theta_{nl} , 0))$$

- It is worth noting that light vector used here goes from object to light for the purposes of calculation.

- More shiny objects use a greater m value creating sharper specular highlights. It is also worth noting that we clamp the cosine value to a minimum of zero to avoid negative values.

- We haven't talked about ambient and emissive because they are simply added and don't need calculating functions for movement of object or view. A good way to understand all this is looking at the picture below.

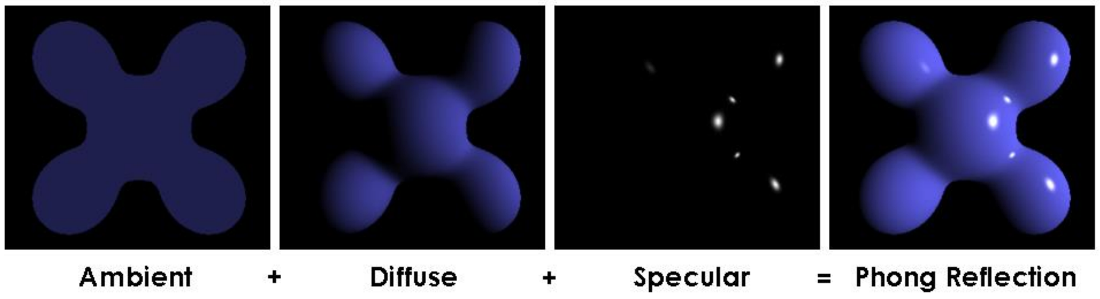

Fig. 4: Combining Ambient, Diffuse and Specular to create Blinn-Phong material.

- In fact the discussion on the link for Fig.4 provided in list of figures is pretty informative and relevant to what has been discussed till now.

- Textures, Reflection and Refraction:

- Texture Mapping: Many real-world objects don't have single colours and are tough to represent in a believable manner using functions. So a bitmap image is mapped to a surface using a u-v coordinate system varying from 0,0 to 1,1.

- UVs can be attached to vertices of a geometry. Usually this is done in 3D software that provide simple mechanisms like plane, cube, sphere or cylinder projection mapping and complex mechanisms, which come in handy for organic models, like peel and pelt mapping.

-Some manual work might be required for some models like specifying seams (where texture can be discontinuous) and rearranging uvs. The final mapping is generally checked against a checkerboard pattern to see if it's distorted.

-A good UV map has evenly sized squares like in the picture below. Although manual effort needs to be put to correct the grey square near the rear wheels which has a corner stretched to the door.

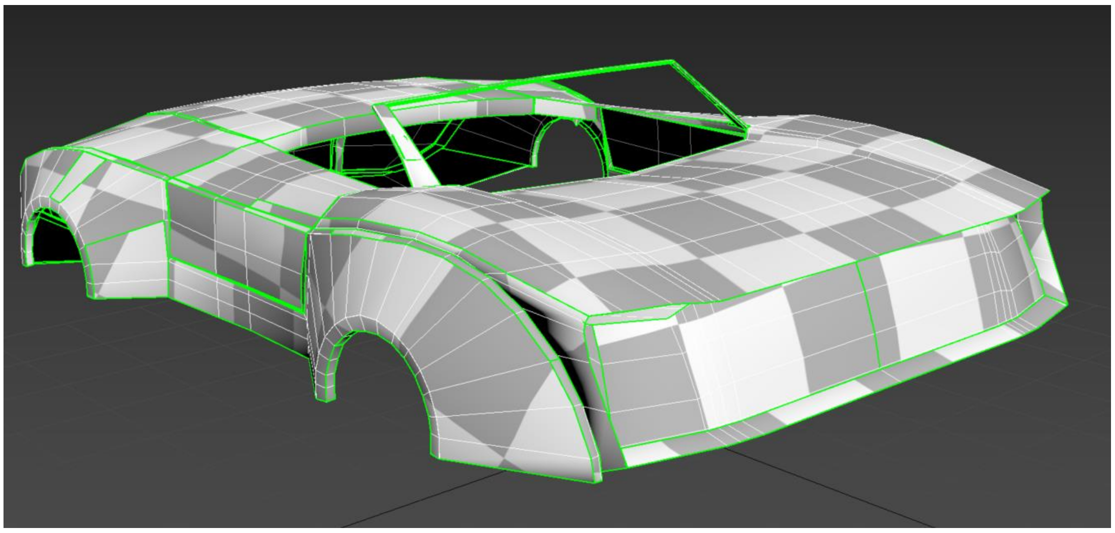

Fig. 5: UVs created by Box Projection followed by Quick Peel without user-defined seams on 3DS Max.

- The textures work well when the objects size, in terms of pixels, on the screen is comparable to the textures size. To differentiate a pixel in the texture we call it a texel. The ideal texel to pixel ratio is 1:1.

- When too many pixels are in a pixel the texture works as if it is zoomed leading to aliasing and the phenomenon is called magnification. When too many texels are in one pixel, colors can seem wrongly positioned as the pixel picks color of a random texel and the phenomenon is called minification.

-Notice how the closer part of the wall on the figure on the next page seems like a zoomed in image, the middle seems the right amount of sharp and the farther one seems smoothed.

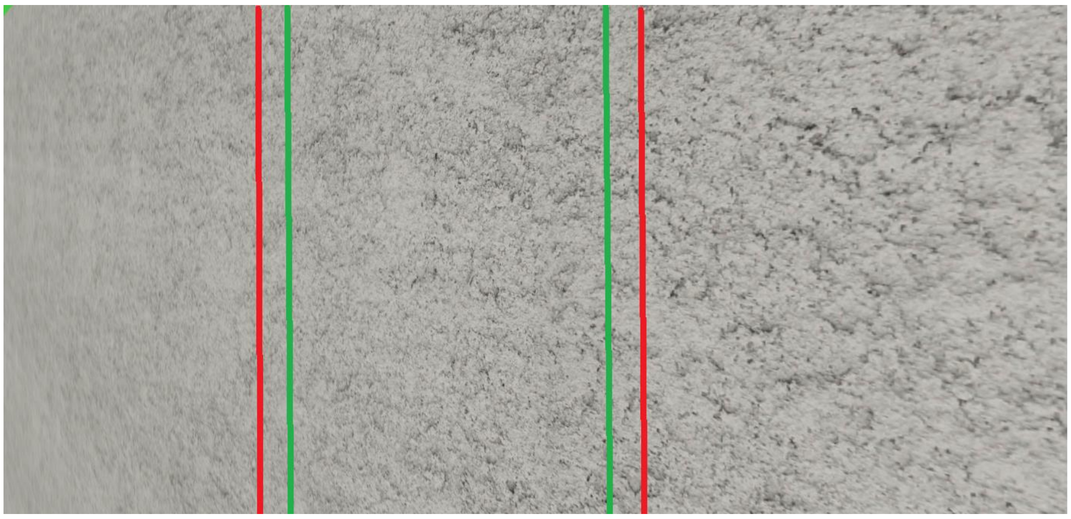

Fig. 6: Magnification and Minification on wall texture

- While the wall far, inaccurately blurred seems fine, there are some cases where blurring is highly undesirable like white strips on a road.

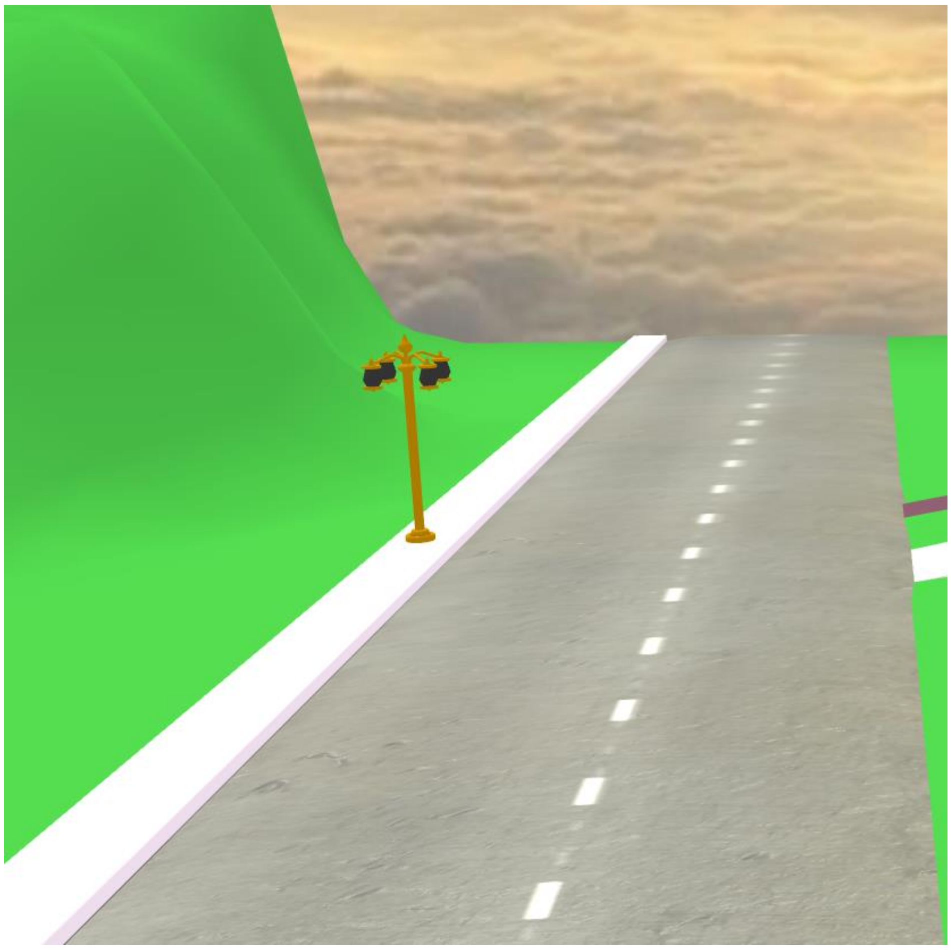

Fig. 7: Minification on road texture

- This is solved by texture filtering which samples surrounding pixels to select colour. Anisotropic filtering improves this by giving preference to sample over x or y. Like in the image above the white strips would benefit by sampling along the strip.

-Another approach is Mip-mapping which stores a stack of low resolution textures of a texture like (256 x 256 + 128 x 128 + 64 x 64 …. 1 x 1). The idea is for far distances, a low resolution texture will result in less texels to select from or filter.

- Skyboxes are a good way of adding good looking sky to scenes using just 6 images mapped to a cube. Since we can't see the edges of the cube the illusion is perfect.

-The skybox is made to move with the camera so that the sky appears still. Care has to be taken that nothing intersects with the skybox and the camera's visible volume includes the skybox for the illusion to be effective.

- Environment irradiance mapping is another effect that uses cube-maps. The idea is that the contributions of the scene for an object per-pixel can be pre-calculated and stored for static objects.

- Cube-maps are also used to do reflection and refraction mapping. This is done by making the object invisible and then rendering the scene to a cube camera (or multiple cameras). This is useful for dynamic reflections like those in racing games.

- This rendered image is then applied to the object as a texture and the scene is rendered with the object. For refraction, the bending of light is accounted for using Snell's Laws, $n_1 \times \sin\theta_i = n_2 \times \sin\theta_r$ where n represents the refractive indices and theta is the angle of incidence and refraction.

- Reflection and refraction mapping is inaccurate but good enough for real-time applications. Textures are also used to achieve various effects.

- Bump map uses grayscale image to make the object look raised by making the whiter areas in the bump map look brighter. Another scheme called normal map is used more often and works by storing normals per pixel by encoding x to r, y to g and z to b channel.

- Displacement maps moves the pixel position using the whiteness of the image used. Specular maps are used to represent real objects that don’t have uniform specularity. Alpha maps are used to blend objects. Normal and bump mapping fail at shallow angles where details don’t block each other. For this relief, height-field and parallax mapping are used which give better results with their own set of drawbacks. Light mapping is a technique used to store lighting calculations for static objects also called baking light.

- GPU Pipeline:

- The GPU pipeline can vary from device to device. Any model starts its journey as model-space coordinates which are then brought into world-space. These coordinates are then oriented with respect to the camera. These processes happen at the application stage.

- These coordinates are then sent as triangles. If a triangle is partially in the camera, the portion outside is removed and new triangles are created. The resulting coordinates are normalized into a box varying from -1,-1,-1 to 1, 1, 1. These coordinates are called Normalized Device Coordinates (NDC).

- The coordinates obtained now are used to execute programs called vertex shaders. The calculations are sent down the pipeline and interpolated by the inbuilt hardware. These calculations are then used by the fragment shader to give the final colour.

- Hardware used to have shaders built-in, but most devices now support programmable shaders that have been used for endless amount of effects like toon shading, fractal noise based materials etc.

- Another term relevant to the pipeline is the bottleneck. Often, we talk of performance as frames per second, but from a developer's perspective a better approach is milliseconds per frame. We want to know how much milliseconds each task contributes to a frame. It is something like speed or time.

- Many tasks are order dependent, so some tasks have to wait for others like in an assembly line. In such cases the slowest task decides the speed of the pipeline and is referred to as the bottleneck. With multi-core architectures some CPU tasks can be executed in parallel for more efficiency.

- Z-Buffer and Shadows:

- Z-Buffer is used to handle overlapping objects. Every fragment of a triangle being processed has a z-depth value. Every pixel allows drawing of only fragments that have z-depth higher than that stored at the pixel/fragment location in screen space.

-The depths are distance values from the camera. If z-depth of two objects at a pixel are equal or nearly equal it leads to a problem called z-fighting. This happens due to the NDC coordinates being same and hence both objects seem to be fighting for the same place. This problem is generally solved by removing the overlapping part.

- The image on the next page shows two planes at same height showing z-fighting.

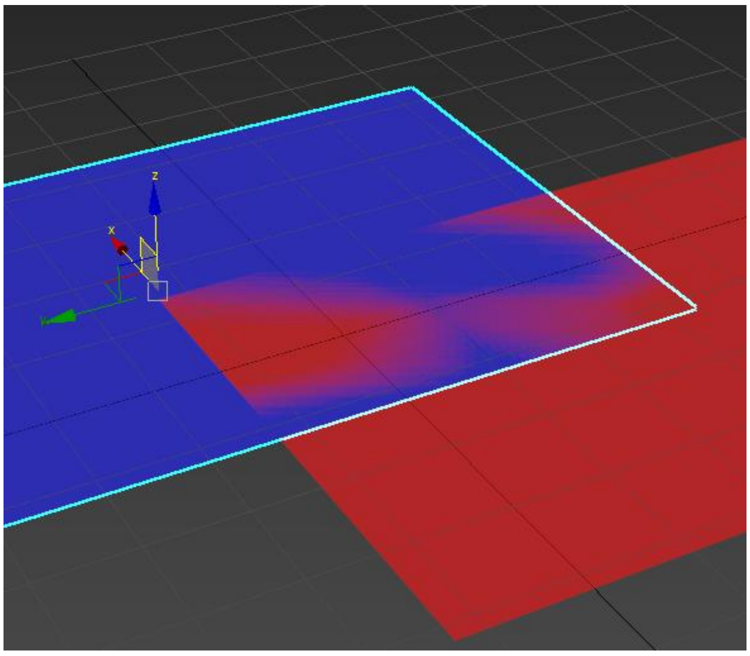

Fig. 8: Z-Fighting

- A good order to draw objects is from front to back as back objects will be ignored by z-buffer during their draw-calls saving computations. In back to front order, the fragment behind is put in the z-buffer and then the fragment in front replaces it which is a waste of time.

- Shadows also use a z-buffer type mechanism in the technique called shadow mapping. A z- buffer is created from the light’s view and the information is used to identify, which parts of the scene are not visible.

- But the problem with this technique is because distances can be slightly different in camera and light view, some areas not in shadow will be rendered as in shadow. A shadow bias is introduced to solve this which introduces a new problem where objects seem detached to their shadow. This is called Peter-Panning.

- A little more work needs to be done for soft shadows as real shadows don’t seem to have crisp boundaries when they are at a certain distance from the shadow. For this Percentage-Closer Soft Shadows (PCSS) technique is used.

- Shader Programming:

  - This is the programmable part of the pipeline which gives developers the power of the GPU. Shaders are mainly vertex and fragment shader, with newer shader models supporting a geometry shader feature. For all vertices, their shader programs are executed and fragments are generated which have their own shader programs.

  - But unlike CPU where you have control over the order of execution of tasks, the GPU programs are like several programs being executed on efficient data structures with almost no control on when which program is executed.

  - The vertex shader is used to specify shading normal and colours for vertices. More things can be specified and the outputs are interpolated for fragments. Every vertex shader must give fragment coordinates as an output. The geometry shader is used to modify vertices and faces which is helpful in situations like deforming objects.

  - The interpolated values are used by fragment shaders and generally functions are used to calculate fragment colour which is outputted by every fragment shader.

  - Shaders have two type of inputs, variables defined in the program and uniforms that have to be predefined and are passed on from application to shader. Normals and colours are examples of commonly used uniforms.

- Quaternions:

  - Matrices take up more space and are inefficient to calculate. Quaternions are a more useful approach to represent rotations and consists of only four values. Quaternions are said to represent movement of a point on the surface of a sphere.

- Euler Angles is another rotation mechanism which rotates x, y, z axes by angles but has a problem called gimble lock in which when two axes coincide a degree of freedom is lost. Quaternions don't have this problem.

- Ray-Tracing and Path-Tracing: These methods represent the real world lighting model more accurately but are too expensive for real-time applications.

- Ray-Tracing: A number of rays are shot from the camera and the paths which bounce and reach the light are traced and contributions are updated. This also directly deals with creating shadows, reflections and refraction. A good example is an image on the next page, whose author has been kind enough to allow uncredited usage of the picture as long as no laws are violated.

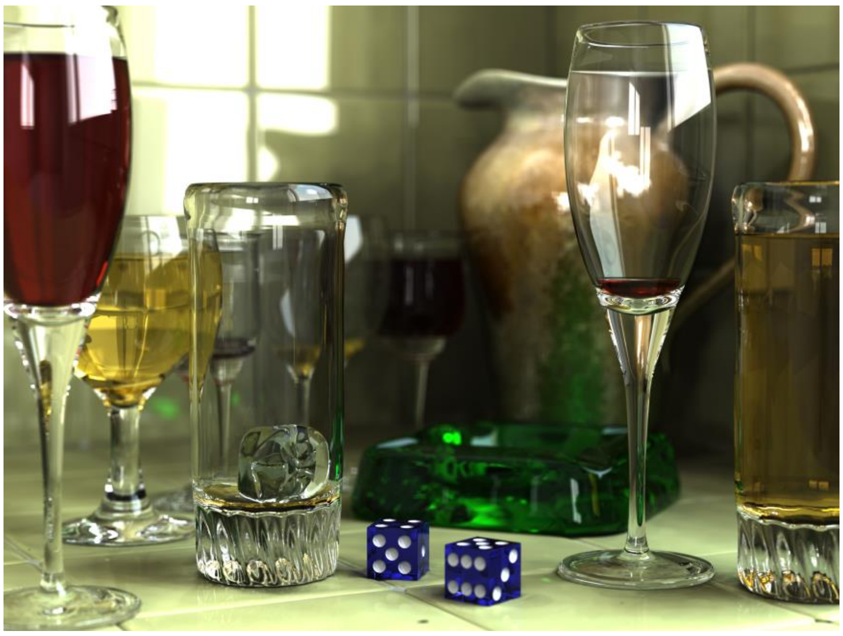
Fig. 9: Realistic Glass by Ray-Tracing

- Path-Tracing: What Ray-Tracing doesn't take care of is light bouncing, reflecting and refracting and then hitting the camera or other objects so in path-tracing we trace rays shot from the light. This gives almost real results but takes a lot of time.

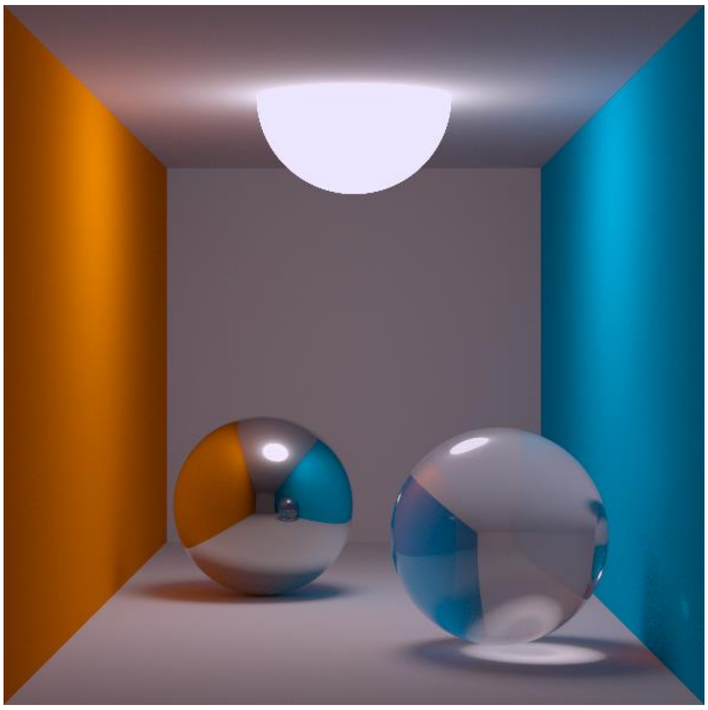
Fig. 10: Path-Tracing

- Transparency:

- This is still kind of an unsolved problem. THREE.js uses the concept of alpha blending to solve transparency. Colour vectors have an alpha component other than r, g, b to represent the contribution to a pixel or opacity.

- For an object in front of an object, we call the front one as a source (s) and the one behind as destination (d). Calculations for alpha blending are as follows:

$$\text{Colour} = \alpha_s * \text{Colour}_s + (1 - \alpha_{s)}) * \text{Colour}_d$$

- This scheme generally renders opaque objects first and then transparent objects in back to front manner for proper blending. If a transparent object in front is drawn, objects behind would never make it on the z-buffer.

- This works well for many situations but can have problems for interpenetrating objects and self-blending objects. A method called depth peeling solves this at high performance cost by finding surfaces next closest to the camera iteratively, starting from the closest.

- Another simple and fast method is screen-door transparency in which pixels are alternated between the colours of two objects. But this gives problems for ` more than two objects and is only good for 50 – 50 object contributions.

- Collision Detection and Response:

-This involves detecting two objects intersecting, using shapes that are simpler than the actual objects being used. Very often ray-casting is used to check intersections using rays being spawned with some criterion.

- Second part of the problem is to decide what action to perform after the collisions of a type are found, which can involve using more parameters. Walking games use a simple cylinder around the character to detect collision with walls, ground and terrain.

- Going through all objects can be computationally expensive so one mechanism used is Binary Space Partitioning (BSP) which works on data structures called BSP Trees for efficient object traversal.

- Ambient Occlusion: This effect handles the phenomenon where object casts shadows on themselves. This is done usually by textures or screen-space calculations.

- Motion Blur: This effect gives an illusion of higher frame-rates and fast speed by interpolating data from fewer frames.

- Anti-Aliasing: This effect uses filters to smooth out jagged lines. This effect comes with various kinds of techniques
/

- V-Sync: What V-Sync does is if the frame is partially painted but the screen refresh rate hasn’t reached the point where it updates the screen, no frames are outputted until the next refresh, in simple words keeping the frame rate as fast as the refresh rate of the screen. But the calculations and freezing of frames can lead to another problem called input lag, where in the input seems to update the application at a delay. So, the rule is turn on V-Sync only if input lag is acceptable or you are getting screen tearing.

- Culling: Removing faces that won’t be visible. A largely used form is back-face culling. Face vertices are in an array in a counter-clockwise order. So clock-wise phases are treated as back-face and removed before calculation to save time.

- Modelling:

    - The process of creating models through software involves knowing all kinds of tool supported by the software being used. For making models we usually start with planes, spheres, cylinders etc and add edges and faces reaching a final model. This is a top-down approach. Many times we use a bottom-up approach, which involves creating the details of the model and connecting them appropriately to get the final product.

    - For the models in this project, quads have been used, which act as two triangular faces while rendering. For creating the structure a lot of edge tools available in 3DS Max have been used.

    - The Cut tool allows creating a line segment starting from an edge or a vertex and ending at an edge or vertex, intersecting with all the faces that come in-between, creating new edges and all the lines that come in-between, creating new vertices.

    - The Chamfer tool creates more edges out of an edge selection and come in-handy when smoothing a surface or creating curved-bevel edges from sharp edges.

- The Connect tool draws edges to connect the edges selected, and the edges drawn only connect edges on the same face.

- Polygons and edges both support the Extrude tool, where extruding edges creates a new face connecting corresponding vertices of newly formed edge and old edge via more edges and extruding polygons connects vertices of the polygons with edges and edges of the polygon with polygons.

- There is a relax tool used to move vertices and edges to allow for smoother looking surfaces. There are smoothing groups which allow for separate shading normal for separate set of polygons.

- A lot of modifiers acts as aids to creating models like subdivision modifier creates more intermediate edges depending in the no. of iterations parameter, more giving a smoother look.

- A lot of tools are used with a lot of parameters and using them makes for a better understanding than just reading about them so we won't talk more about them.

1.2. **Motivation**: THREE.js provides a lot of basic primitives and examples for different aspects of graphics programming and hence is a good platform when it comes to experimenting with different techniques.

1.3. **Aim of the proposed Work**: To create an in-browser interactive 3D experience that delivers a good experience with appreciable performance.

1.4. **Objective(s) of the proposed work**:

- To show common widely used graphic programming techniques.
- To provide an element of interactivity.

2. Literature Survey

Shadows:

- Shadow mapping has its limitations but is computationally inexpensive in contrast to volumetric methods. While the undesirable effects are still there, two methods have brought us very close to perceptually good looking shadows.

- First is Percentage-Close Filtering, where shadow boundaries harden as the objects get closer to the surface on which the shadow is being cast. This method simulates the umbra and penumbra phenomenon also, which refers to the middle of the shadow being evenly dark and the edges of the shadow gradually changing from dark to light.

- Since this method creates, what is called, soft-shadows it is also called Percentage-Close Soft Shadow or PCSS Technique.

- Second is cascaded shadow maps. The issues with shadow mapping only aggravate when they look different at different distances. A way this is solved is using cascaded shadow maps, which divides the view frustum into several smaller ones for different distances.

- The idea is using higher filtering for closer distances and lower for farther distances. It is computationally expensive to use the filtering used at closer distances for farther distances and doing the opposite is not good for shadows at closer distances.

Smoothing:

- Due to limits on resolution of screen, textures and precision of calculations we get the undesirable effect called jaggies that are staircase like lines. In motion these are called crawlies.

- One of the fast, widely used techniques to solve jaggies is FXAA. It is a low performance cost – high quality improvement technique, this technique can also be mixed with post-processing in some cases like motion blur.

- But other methods like CMAA, MSAA and TXAA give better results. TXAA is what NVIDIA calls a film-style smoothing technique which also deals with crawlies. It is the most expensive of all mentioned but the top of the line GPUs can support it for many games and so it find its way in many applications.

- MSAA is also a widely used technique which is generally limited to edges to save resources as that’s where aliasing is mainly a problem. CMAA is found in lesser games but its debated that it’s a cheaper technique and is good for games in an artistic sense thereby being better for quality and performance. These two fall between FXAA and TXAA in terms of quality and performance.

- A lot of code for smoothing algorithms is available and information on these algorithms is mainly needed for selecting the ones to suit the graphic style and trying to add to a technique or develop an entire new technique.

Transparency:

- Alpha-Blending is a widely used technique for transparency and even THREE.js uses it as the default way of dealing with transparency. But it is computationally expensive due to the formula being needed to be applied for all transparency pixels for every frame in back to front order. A little power is saved by using textures with pre-multiplied alpha values.

- Alpha-Blending is an order-dependent transparency technique. A lot of research is being done in order-independent transparency as it seems to have the potential for better speeds and perceptual correctness due to its nature.

- One method is depth-peeling, which iteratively looks for the surface closest to camera, by removing the closest layer and repeating the search. These layers can then be blended in any order using alpha values. This is expensive but works always and there is scope for further research with hardware and software getting better.

- Another less computationally expensive approach being suggested is using screen-door transparency in a way that looks perceptually fine by correctly averaging out layers. A paper in the references suggests doing this by using properly generated pixel masks. This also is order-independent and so has similar advantages.

3. Overview of the Proposed System

3.1. Introduction: Just like OpenGL ES, WebGL is a programming API which executes like a state machine.

- The program initializes with init( ) method. All things in the scene can be introduced in init( ) or init( ) can call functions which can call more.

- The function render( ) is used to specify the render call, create effects like reflections using additional render passes. etc. The function animate( ) calls render( ) method again and again and it calls itself to move the animation forward. Animate( ) can be used to change the scene using user input as done in this project.

3.2. Architecture for the Proposed System

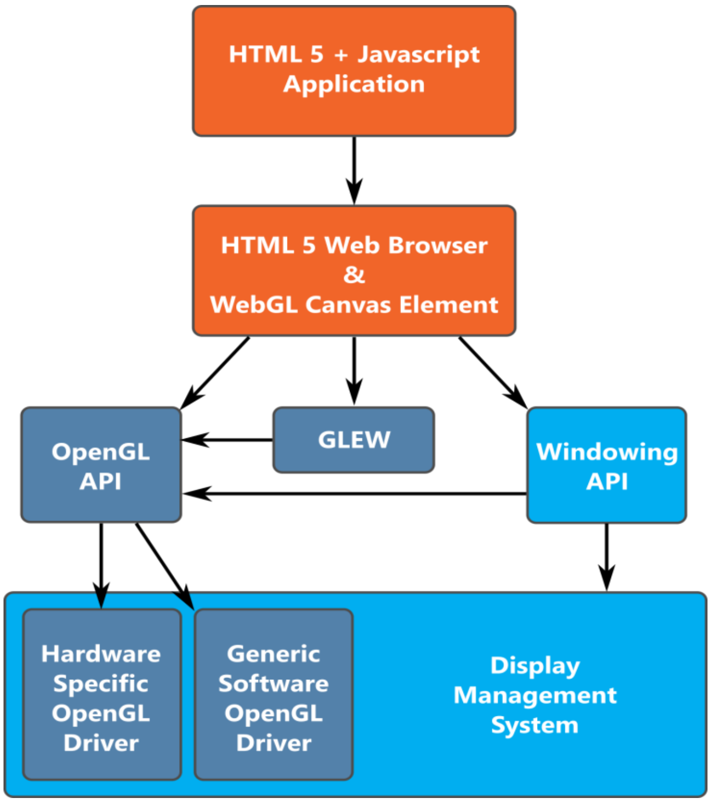
HTML 5 + Javascript Application
HTML 5 Web Browser & WebGL Canvas Element
OpenGL API
GLEW
Windowing API
Hardware Specific OpenGL Driver
Generic Software OpenGL Driver
Display Management System

Fig. 11: WebGL on an ideal OpenGL Device

- OpenGL is largely being used for cross-platform applications and on the previous page is a generic WebGL architecture for an ideal OpenGL Device.
It is worth noting that Windows has DirectX Drivers instead of OpenGL Drivers and so it uses ANGLE which converts OpenGL API calls to DirectX API calls. The source of the image is a good place to get to know more about the significance of this conversion.

4. Proposed System Analysis and Design

4.1. Requirement Analysis

4.1.1. Functional Requirements

4.1.1.1. Product Perspective: The demo should celebrate the commonly used graphic techniques in a user appealing manner and provide a taste of the capabilities of THREE.js and WebGL to people interested in exploring the field of graphics programming.

4.1.1.2. Product features:

- Utilizes current-gen graphic techniques.

4.1.1.3. User characteristics:

- New to graphic programming or amateur.
- Interested in Web-based 3D applications.

4.1.1.4. User Requirements

- Interactivity using keyboard.

4.1.2. Non Functional Requirements

4.1.2.1. Product Requirements

4.1.2.1.1. Efficiency: GUI support for reducing quality to support demo at real-time performance on user end.

4.1.2.1.2. Reliability: Bug-free experience to avoid users repeating same mistakes.

4.1.2.1.3. Portability: Focus on PC/Laptop-based rendering

4.1.2.1.4. Usability: GUI support for extending compatibility.

4.1.3. Engineering Standard Requirements

- Social: A lot of developers contribute to the THREE.js GitHub repository or their own separate repository. GitHub is a great place to share code to help all kinds of developers.

- Ethical: Such demos require different kind of skills. So a lot of content, especially textures have been imported by third-parties and so it is important to give credit where credit is due.

- Legality: 3DS Max Student Version has been used to create many models in this demo and so this demo will be used for educational purposes only.

- Inspect-ability: The code will be readily available for others to run locally and use GUI options provided in the demo as well as graphic debugging tools already provided by Firefox and Chrome to experiment with the demo.

4.1.4. System Requirements

4.1.4.1. H/W Requirements(details about Application Specific Hardware)

- Currently focusing on Hardware that supports Shader Model 4.0
- A dedicated GPU system is recommended for appreciable performance.

4.1.4.2. S/W Requirements(details about Application Specific Software)

- A system that supports WebGL API.
- A browser with WebGL support, preferably Firefox or Chrome

5. Results and Discussion

- Reflective mapping is a decent trick to create metal like materials which is more believable for rough surfaces.

- Simple mathematical functions can be used to create believable car physics.

- While playing with pre-defined primitives is helpful, applying radiometry and colorimetry concepts give more believable results.

- Software and hardware are moving towards better quality rendering at interactive rates but manual tricks to improve performance and quality are not going out of importance any time soon.

6. Conclusion, Limitations and Scope for Future Work

While the project is a good example of different effects, there is always room for improvement with programming that takes advantage of multi-core CPUs and acceleration algorithms. A better understanding of all aspects would lead to a better code. There is just so much variety of phenomenon whose simplifications make for whole subjects and hence, more the time spent learning, better will be the implementation. Even for someone who knows such techniques and subjects well, there is still scope for more. A line from the book Real-Time Rendering really bursts the bubble by stating “One of the great myths concerning computers is that one day we will have enough processing power.” We are still making simplifications to account for innumerable rays of light and we use believable tricks to get closer to the right result, more tricks for scenes with more objects. But it is easy for something to not be accounted for. Doing realism at interactive rates is currently only achievable by tricks and has limits to complexity that can be handled and so the field is moving towards better hardware and software with time.

GitHub Repositories(For sample code and header files)

1. THREE.js by mr. doob
   https://github.com/mrdoob/three.js/wiki

2. AlteredQualia
   https://github.com/alteredq

3. stemkoski
   https://github.com/stemkoski/stemkoski.github.com

Resources:

1. 3DS Max script to export geometry, vertex normals and uv coordinates to a JSON file(this file can be accessed by THREE.js OBJ Loader)
https://github.com/timoxley/threejs/blob/master/utils/exporters/max/ThreeJSExporter.ms

2. Skyboxes
https://i.pinimg.com/originals/64/37/cb/6437cb09b1d0fe9ff2159118157e1a84.png?epik=0UTnDE_IW37J1 (Day)
https://i.pinimg.com/originals/64/37/cb/6437cb09b1d0fe9ff2159118157e1a84.png?epik=0UTnDE_IW37J1 (Night)

2. Car Wooden Panel Texture:
https://wallpaperjam.com/view/red-textures-wood-texture-56098

3. Car Seat - Leather Texture (CC-BY 3.0)
https://opengameart.org/content/fabric-leather-seamless-texture-with-normalmap

4. Road Textures:
https://www.textures.com/download/roads0060/35211
http://www.textures.com/download/3dscans0022/126610

5. Wheel Texture:

https://www.textures.com/download/wheels0085/30964

6. Car Floor - Carpet Texture

https://www.123rf.com/photo_68718989_industrial-black-vinyl-carpet-coil-pattern-car-floor-mat-texture-close-up-anti-slippery-surface-viny.html

7. Car Indoor - Leather Texture

https://www.textures.com/download/substance0022/126941

6. Grass and Cliff Textures

Thanks to Christian Femmer aka Duion under CC-BY 3.0 (License)

7. Door and Frame - Wood Texture

https://www.textures.com/download/woodfine0031/31730

8. House Wall - Brick Texture

https://www.textures.com/download/3dscans0115/132042

9. House Path - Soil Texture

https://www.textures.com/download/3dscans0293/132865

10. Garage Ground - Concrete Texture

https://www.textures.com/download/concretefloors0065/44814

11. Garage Wall - Wall Texture

http://polygonblog.com.www32.zoner-asiakas.fi/wp-content/uploads/2010/04/concrete-wall-texture-high-res.jpg

LIST OF FIGURES

Title **Page No.**

Fig. 1 - Comparison: No Gamma Correction vs Gamma Correction

2.

(By Wolfire Blog - http://cdn.wolfire.com/blog/gamma/compare1.jpg)

Fig. 2 - Different Colour Spaces

3.

(By BenRG and cmglee - http://commons.wikimedia.org/wiki/File:CIE1931xy_blank.svg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=32158329)

Fig. 3 - Darth Vader Mask in 3DS Max

5.

(Some of my work under 3DS Max Student License)

Fig. 4 - Combining Ambient, Diffuse and Specular to create Blinn-Phong material.

11.

(https://computergraphics.stackexchange.com/questions/1513/how-physically-based-is-the-diffuse-and-specular-distinction)

Fig. 5 - UVs created by Box Projection followed by Quick Peel without user-defined seams on 3DS Max.

12.

(Intermediate version of model used for this project)

Fig. 6 - Magnification and Minification on wall texture. 13.

(Intermediate version of material used for this project, Resource 4)

Fig. 7 - Minification on road texture 13.

(Intermediate version of material used for this project, Resource 3)

Fig. 8 - Z-Fighting 17.

(Two simple planes in 3DS Max)

Fig. 9 - Realistic Glass by Ray-Tracing 19.

(Author allows for free distribution of image unless any laws are violated https://commons.wikimedia.org/wiki/File:Glasses_800_edit.png)

Fig. 10 - Path-Tracing 20.

ThKa (Thomas Kabir) (https://commons.wikimedia.org/wiki/File:Box_-_Path_Tracing_High.png), „Box - Path Tracing High“, https://creativecommons.org/licenses/by-sa/2.0/de/legalcode)

Fig. 11 - WebGL on an ideal OpenGL Device 27.

("Ideal" WebGL Application / OpenGL / OS interactions. http://www.cs.uregina.ca/Links/class-info/315/WebGL/Lab1/)

LIST OF ABBREVIATIONS

Abbreviation	Expansion
CPU	Central Processing Unit
GPU	Graphics Processing Unit
HDR	High Dynamic Range
RGB	Red Green Blue
sRGB	Standard RGB
CMYK	Cyan Magenta Yellow Key
BRDF	Bidirectional Reflectance Distribution Function
NDC	Normalized Device Coordinates
PCSS	Percentage Close Soft Shadows
CSM	Cascaded Shadow Mapping
BSP	Binary Space Partitioning
FXAA	Fast-Approximate Anti-Aliasing
CMAA	Conservative Morphological Anti-Aliasing
MSAA	Multisampling Anti-aliasing
TXAA	Temporal Anti-aliasing
CC-BY 3.0	Creative Commons Attribution 3.0 License (In basic terms it's freely using someone's creative work and following terms of attribution specified by the author if any. For more information visit the site below https://creativecommons.org/licenses/by/3.0/)

REFERENCES

1. UDACITY CS291: Interactive 3D Rendering
https://classroom.udacity.com/courses/cs291

2. Real-Time Rendering Third Edition by Tomas Akenine-Moller, Eric Haines and Naty Hoffman. They keep their work updated at:
http://www.realtimerendering.com/

3. GLSL Reference Guide
http://mew.cx/glsl_quickref.pdf

4. The Book of Shaders
https://thebookofshaders.com/

5. Cascaded Shadow Maps
Dimitrov, R. (2007). Cascaded shadow maps. *Developer Documentation, NVIDIA Corp*.

6. Percentage-Closer Soft Shadows
Fernando, R. (2005, July). Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches* (p. 35). ACM.

7. Depth-Peeling
Everitt, C. (2001). Interactive order-independent transparency. *White paper, nVIDIA*, *2*(6), 7.

8. Screen-Door Transparency – Using Pixel Masks
Mulder, J. D., Groen, F. C., & van Wijk, J. J. (1998, October). Pixel masks for screen-door transparency. In *Proceedings of the conference on Visualization'98* (pp. 351-358). IEEE Computer Society Press.

9. FXAA
https://docs.nvidia.com/gameworks/content/gameworkslibrary/graphicssamples/opengl_samples/fxaa.htm

10. CMAA vs Others
https://software.intel.com/en-us/articles/conservative-morphological-anti-aliasing-cmaa-update

11. Shadershop(Used to visualize curves used for simplified car physics)
http://www.cdglabs.org/Shadershop/