International Journal of Computer Applications (0975 – 8887)
Volume 115 – No. 17, April 2015

3D Modeling and Simulation using Image Stitching

Sean N. Braganza
Computer Department
K. J. Somaiya College of Engineering, Mumbai, India

ShubhamR.Langer
Computer Department
K. J. Somaiya College of Engineering,Mumbai, India

Sushant R. Gaikwad
Computer Department
K. J. Somaiya College of Engineering, Mumbai, India

Pallavi G.Bhoite
Computer Department
K. J. Somaiya College of Engineering
Mumbai, India

ABSTRACT

In recent years, 3D modeling has played an increasingly significant role in various fields, including motion pictures, video game industry, earth science and medical industry.Yet, 3D modeling continues to be a complex and tedious process which involves the use of various high-end devices such as 3D scanners which in turn require great expertise while working with them.A series of algorithms and several calculations produces “scans” that have to be merged to create a three dimensional true to life representation of the model. This makes the entire process cumbersome, especially when modeling interior scenes for purposes of simulation. This paper introduces a cost-effective modeling method for the same which is least complex, user friendly and involves the concept of Image Stitching and 3D graphics rendering software while also allowing simulation of user movement within the created scene on a computer. The paper will demonstrate that the proposed system is more effective and efficient at the same time.

General Terms

Image stitching, Key-points, Simulation, 3D modeling, Blender, UV Mapping, SIFT, RANSAC, OpenGL, API

Keywords

3D Modeling, OpenGL, Interior Modeling, Blender, UV Mapping, SIFT, RANSAC, Feature Detection, Homography Estimation, Wavefront, Mesh, Model, Computer Graphics, Assimp, GLFS, GLEW, SOIL

1. INTRODUCTION

In Computer Graphics, the process of developing a mathematical representation of any 3D surface object using specialized software is called 3D modeling. It is used in various industries like films, animations, gaming, interior designing and architecture and has a wide range of applications.

Simulating movement within a room requiresthe creation of a detailed 3D model of the room concerned. This would involve, capturing various characteristics of the room using the different modeling methods before simulating movement. This in turn involves connecting points in 3D space by line segments to form polygonal mesh. For those without access to required equipment, skill or expertise, the entire task could prove to be complex and infeasible especially in cases where only a rough and approximate model of the room is required.However, the approach suggested in this paper aims to simplify the task by doing away with tedious modeling methods and using instead, the concept of image stitching to obtain realistic images of the room which would then be mapped to computer aided 3Dgraphical structures representative of the room. Simulation will then follow.

The first step involves capturing multiple photos of the room to be modeled to obtain apanoramic view of the room. This means physically setting up the camera, configuring it to capture all photos identically, and then taking the sequence of photos. The end result is a set of images which encompasses the entire field of view, where all are taken from virtually the same point of perspective.

The next step is the use of Blender, which is an open source 3D computer graphics software, for creating the model of a room to which the stitched images from the first step will be mapped. Note that henceforth, the room referred to in this paper is a simple four walled room with a ceiling and floor. Hence, a 3D cuboid with dimensions replicating the walls, ceiling and floor of the room is created.Using UV mapping stitched images of the walls are mapped to this computer generated 3D model. The output of this will be a room with the images mapped on to its walls.

The final part involves simulating user movement within the room. This involves the use of Open Graphics Library (OpenGL), a graphics API (Application Programming Interface) that allows the rendering of 2D and 3D graphics. The model obtained from the previous stage is imported into an OpenGL application that has been programmed to accept user input by means of the mouse and keyboard. The final result would be an application consisting of the 3D room wherein simulation of movement around and within will be possible.

2. IMAGE STITCHING

Image Stitching is a process of combining multiple photographic images with overlapping regions to produce a segmented panorama or a high resolution image [1].Sometimes when capturing an image of an object, only a partial image of the object is obtained especially when the size of the object is quite large. However if two or more images withoverlapping regions are obtained, they can be merged on the basis of the overlapping portions to obtain a picture of greater size and resolution that what would have been obtained from a single picture of the object. Based on this concept, images of the room to be modeled are captured in the essence that only the walls are photographed. However, each wall will be photographed in two halves.

Image stitching algorithms involves two basic stages, namely, Feature Detection and Homography Estimation based on those detected features.

2.1 Feature Detection

For any object in an image, interesting points from the object can be extracted to provide a "feature description" of the object. This description extracted from an image can then be used to identify the object when attempting to locate the object in an image containing many other objects. For reliable recognition, we need to ensure that the features extracted are detectable even when an image undergoes change in scale, noise and illumination. These points are usually found in high contrast regions of the image. Also the relative positions between the features should not vary from one image to another [2].

There are different algorithms for featuredetection, namely Scale Invariant Feature Transform(SIFT), Speeded Up Robust Features (SURF) and Features from Accelerated Segment Test(FAST).

A comparative study of these algorithms yielded the following findings:

1. The amount of features detected is purely dependent on the type of images used.
2. For highly textured images, SIFT and SURF detected more features.
3. SIFT found more matches then SURF and FAST.
4. The amount of features detected is proportional to the amount of matches [3].

This concluded that SIFT was most appropriate for the proposed system.

2.1.1 Scale-Invariant Feature Transform

The SIFT algorithm proposed by David G. Lowe is an approach for extracting distinctive invariant features from images. It has been successfully applied to a variety of computer vision problems based on feature matching including object recognition, pose estimation and image retrieval among others. The different stages involved in the SIFT algorithm are:

1. Scale-space extrema detection: Using a difference-of-Gaussian function, a scale space is generated for each of the images provided to the algorithm. The images' scale space is then searched to determine points of interest. These points are required to be invariant to scale, orientation and other transformations.

2. Key-point localization: These points of interest function as key-points. Their locations and scale are determined.

3. Orientation assignment: Based on the gradient directions of each key-point, an orientation is assigned. Henceforth, all operations are performed on the location, scale and orientation information of the key-points.

4. Key-point descriptor: Around each key-point, image gradients are measured and transformed into a suitable format. This information is regarded as a descriptor and is used in key-point matching in later stages of the algorithm. This approach has been named the Scale Invariant Feature Transform (SIFT), as it transforms image data into scale invariant coordinates relative to local features. [4]

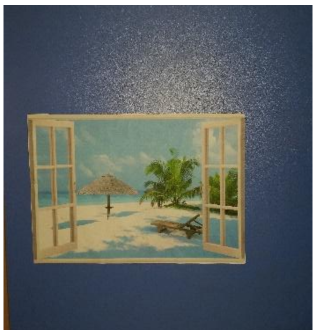
Figure 2.1.1 – Image 1 (Real World Image of a wall)

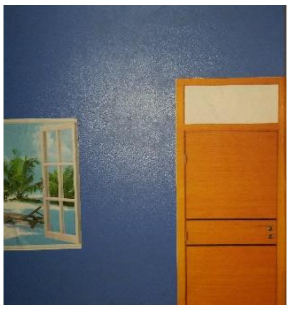
Figure 2.1.2 – Image 2 (Real World Image of the second half of the wall from Figure 2.1 Supplied to SIFT)

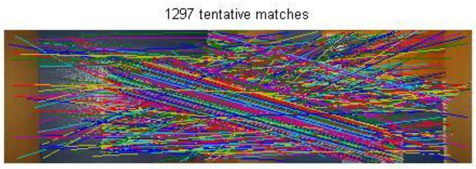

Figure 2.1.3 – Output of SIFT shows keypoints detected between the two images.

2.2 Homography Estimation

2.2.1 Homography

It is possible to represent a 2D point (a, b) in an image as a 3D vector = (a_1, a_2, a_3) where a = a_1/a_2 and b= a_2/a_3. This is the homogeneous representation of a point where the point lies on the projective plane (P2).

A mapping from P2 → P2 is projectivity if and only if there exists a non-singular 3×3 matrix H such that for any point in P 2 represented by vector x it is true that it's mapped point equals Hx. Hence it becomes necessary to calculate the 3*3 Homography matrix H for calculating the x_i that maps with the corresponding x_i'.

Homographies can be estimated by calculating feature correspondences between the images.

The simplest algorithm for solving the homography given a set of point correspondences is Direct Linear Transform(DLT). It solves asset of variables from a set of similarity relations. But the problem with the DLT is that it requires a set of correspondences as input which makes them robust for the cases where the source of noise is in the measurement of the correspondence feature positions. It fails when two features in an image do not correspond to the same feature in the real world. Then the need of another algorithm arises which can robustly differentiate between inliers and outliers so that homography is calculated using only inliers [5].

The best suited algorithm for that is RANdomSAmple Consensus (RANSAC) [6].

2.2.2 Random Sample Consensus

RANSAC is the most widely used for homography estimation due to its robustness. It is an iterative procedure to estimate the parameters of the given model from the available set of data that also contains outliers.

It assumes that the data consists of “inliers” ,i.e. whose distribution is well explained even if it may be subject to noise, and “outliers” whose distribution is not well explained and also do not fit the model.

A random sample of 4 correspondences is chosen and homography H is estimated for these 4 correspondences for a number of iterations. These correspondences are then classified as inliers or outliers that depend on its concurrence with the Homography matrix. The iteration with maximum number of inliers is selected after all the iterations are done. For calculating the Homography matrix, the correspondences which were taken as inliers from that iteration are considered.The major issue with the RANSAC is identifying the correspondences as inliers and outliers. Generally a distance threshold,‘t’ is assigned for classifying them as either.

Another issue with the algorithm is deciding the number of iterations to be run for finding the correspondences. It makes no sense and is also infeasible at the same time if every combination of 4 correspondences are taken. So determining the number of iterations to be run becomes important [7].

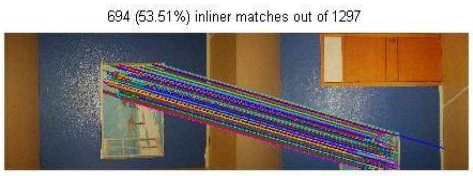

Figure 2.2.1– Output of RANSAC shows number of inlier matches as per estimated Homography Matrix

Using the Scale Invariant Feature Transform(SIFT) and the RANdom Sample Consensus(RANSAC) algorithms in data manipulation software such as MATLAB or OpenCV, the images obtained of each wall of the room are stitched, providing high resolution images of the six interior. The final output will appear similar to Figure 2.2.2.

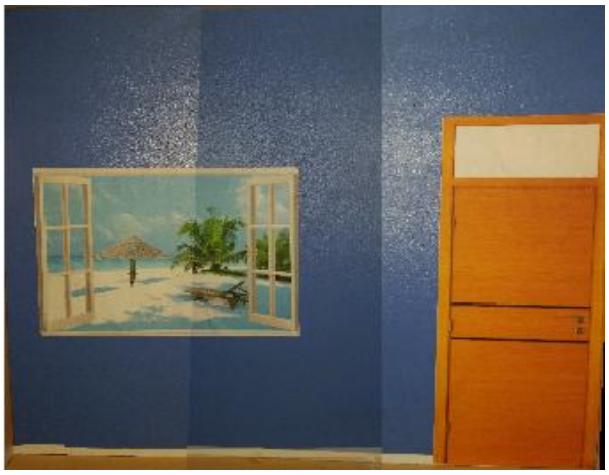

Figure 2.2.2 – Stitched Image. Output of the Image Stitching Stage

3. MODELLING AND UV MAPPING

The procedure followed above results in a stitched image of the real life interior scene that needs to be modeled. After the images have been stitched, the next step is to create a three dimensional model of the room of approximate dimensions.

While there exist several 3D modeling and design applications, Blender is the modeling software of choice.This is facilitated by 3D Modeling Application, Blender, which also aids in mapping images on to the faces of this computer generated model. The end user of the application system however will be required to interact with only a portion of Blender, more specifically, the UV mapping stage.

3.1 Modeling

Blender is a professional free and open-source 3D computer graphics software product used for creating animated films, visual effects, art, 3D printed models, interactive 3D applications and video games. Blender's features include 3D modeling, UV unwrapping, texturing, raster graphics editing, soft body simulation, sculpting.The two primary modes of work that are relevant to the applicationsystem are Object Mode and Edit Mode. Object mode is used to manipulate individual objects as a unit, while Edit mode is used to manipulate the actual object data. Object Mode can be used to move, scale, and rotate entire polygon meshes, and Edit Mode can be used to manipulate the individual vertices of a single mesh. [8]

Blender contains a collection of several meshes that serve as the basis for modeling any 3D object. A mesh, is a collection of vertices, faces and edges that describe the face of a 3D object wherein

- A **vertex** is a single point
- An **edge** is a straight line segment connecting two vertices.
- A **face** is a flat surface enclosed by edges

These meshes can be manipulated by in several aspects inclusive of size, lighting and shape. Relevant to the application is Blender’s ‘Cube’ mesh, consisting of six faces as shown in Figure 3.1.1. This ‘Cube’ mesh is manipulated in size to resemble the dimensional layout of the room.

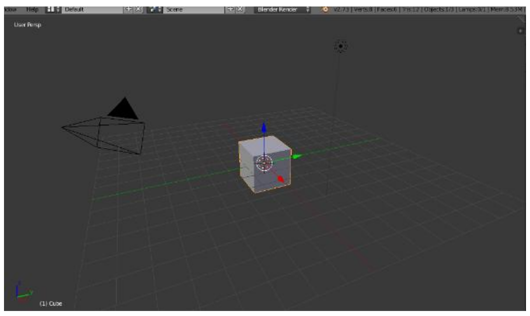

Figure 3.1.1 Cube Mesh

3.2 UV Mapping

Given the cube obtained as a result of the modeling stage and the high resolution, stitched images of the faces of the real world room, the next step involves mapping these images to the faces of the modeled cube.

Image mapping or UV mapping as it referred to, is the process of projecting a texture onto a three dimensional object. ’U’ and ‘V’ denote the axes of the two dimensional texture. UV texturing permits polygons that make up a 3D object to be painted with color from an image. The image is called a UV texture map, but it's just an ordinary image. The UV mapping process involves assigning pixels in the image to surface mappings on the polygon, usually done by "programmatically" copying a triangle shaped piece of the image map and pasting it onto a triangle on the object.UV is the alternative to XY, it only maps into a texture space rather than into the geometric space of the object. But the rendering computation uses the UV texture coordinates to determine how to paint the three-dimensional surface [9].

With respect to the application, textures refer to the real world, high resolution images obtained as a result of the Image Stitching stage. Every point in the UV map corresponds to a vertex in the mesh. The lines joining the UVs correspond to edges in the mesh. Each face in the UV map corresponds to a mesh face.

Each face of a mesh can have many UV Textures. Each UV Texture can have an individual image assigned to it. When unwrapping a face to a UV Texture in the UV/Image Editor, each face of the mesh is automatically assigned four UV coordinates: These coordinates define the way an image or a texture is mapped onto the face. These are 2D coordinates, which is why they're called UV, to distinguish them from XYZ coordinates. These coordinates can be used for rendering or for real-time OpenGL display as well.

Every face in Blender can have a link to a different image. The UV coordinates define how this image is mapped onto the face. This image then can be rendered or displayed in real time. A 3D window has to be in "Face Select" mode to be able to assign Images or change UV coordinates of the active Mesh Object. This allows a face to participate in many UV Textures [10].

Unwrapping the cube results in all its faces lying down on the plane in the form of ‘T’ as shown in Figure 3.2.1. This ‘T’ image is saved as an image file for further processing, preferably in “.png” format due to the high resolution offered and compatibility with other image manipulation software applications.

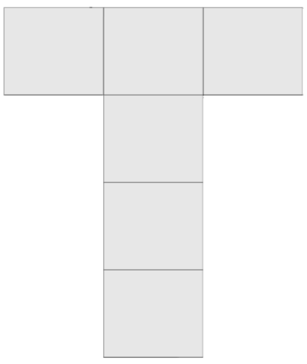

Figure 3.2.1 – An Empty ‘T’ obtained by unwrapping a cube mesh.

The six faces of the ‘T’ are representative of the 4 walls, floor and ceiling of the room. Since this ‘T’ contains information of the cube’s texture maps, irrespective of the images placed in the six faces, simply loading the model in blender will cause the images placed in the ‘T’ to be mapped to the corresponding faces. Using this concept then, the ‘T’ is manipulated externally using any Image manipulation software of choice such as Microsoft Paint. Manipulation here refers to simply pasting the stitched images of the faces of the real world room onto the corresponding faces of the ‘T’.

The output obtained after placing the images obtained from the Image Stitching stage on to the T is depicted in Figure 3.2.2:

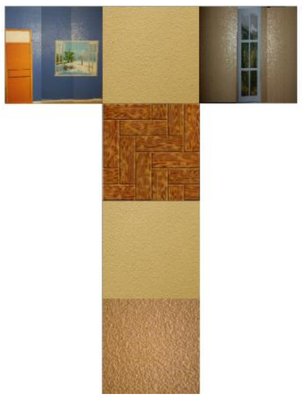

Figure 3.2.2 – Complete ‘T’ obtained as a result of mapping the stitched images onto it

Reloading the cube mesh in blender results in the updated ‘T’ being mapped onto it as depicted in Figure 3.2.3

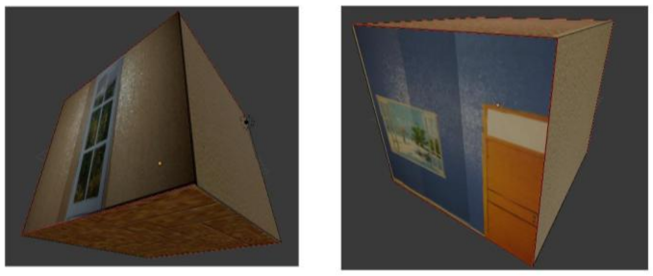

Figure 3.2.3(a)Updated External faces of the cube mesh

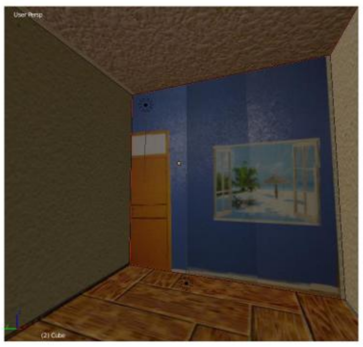

Figure 3.2.3(b)Interior faces of the cube mesh

This mesh serves as the final 3D model rendered for simulation and will henceforth be referred to as the ‘cube model’.

3.3 Exporting the Cube Model

Since the simulation aspect of the application is programmed in OpenGL, it demands that the cube model obtained as a result of the modeling and UV mapping steps be exported in a file format compatible with OpenGL.

A common issue is however that there are dozens of different file formats where each exports the model data in its own unique way. Model formats like the Wavefront.obj only contains model data with minor material information like model colors and diffuse/specular maps, while model formats like the XML-based Collada file format are extremely extensive and contain models, lights, many types of materials, animation data, cameras, complete scene information and much more. The wavefront object format is generally considered to be an easy-to-parse model format [11].

The OBJ file format is a simple data-format that represents 3D geometry alone — namely, the position of each vertex, the UV position of each texture coordinate vertex, vertex normals, and the faces that make each polygon defined as a list of vertices, and texture vertices. Vertices are stored in a counter-clockwise order by default, making explicit declaration of face normals unnecessary. OBJ coordinates have no units, but OBJ files can contain scale information in a human readable comment line.

Materials that describe the visual aspects of the polygons are stored in external .mtl files. The .MTL File Format is a companion file format that describes surface shading (material) properties of objects within one or more .OBJ files. A .OBJ file references one or more .MTL files (called "material libraries"), and from there, references one or more material descriptions by name. [12].

Exporting the cube model in Wavefront .obj file format results in a .obj file and companion .mtl file which references the 'T' that contains mapping information.

4. SIMULATION

Simulation is the process of recreating a real world process. With respect to the application being developed, it refers to the recreation of forward, backward, left, right and strafing movements of a real world person navigating an interior scene. This will be facilitated by means of mouse and keyboard input as done in computer based video games. To facilitate this, and to allow for easy loading of object files and relative simplicity in programming, Open Graphics Library (OpenGL) is used as the programming interface of choice. The purpose of this stage is to create the final build of the application that processes the complete, UV mapped model obtained as a direct result of previous stages and develop an interface to allow for user interactivity.

4.1 Programming in OpenGL

OpenGL is mainly considered an API (an Application Programming Interface) that provides us with a large set of functions that we can use to manipulate graphics and images. However, OpenGL by itself is not an API, but merely a specification.

The OpenGL specification specifies exactly what the result/output of each function should be and how it should perform. It is then up to the developers implementing this specification to come up with a solution of how this function should operate.

4.1.1 Defining the application

The core of OpenGL's functionality resides in libraries written in C language. For this reason, the design of any application requires the use of a compiler such as Microsoft's Visual Studio.

Instrumental to every OpenGL application's design, are the GLFW and GLEW (OpenGL Extension Wrangler) Libraries. GLFW is a library, written in C, specifically targeted at OpenGL providing the bare necessities required for rendering goodies to the screen. It allows us to create an OpenGL context, define window parameters and handle user input which is all that we need. Also, since OpenGL is a standard/specification it is up to the driver manufacturer to implement the specification to a driver that the specific graphics card supports. Since there are many different versions of OpenGL drivers, the location of most of its functions is not known at compile-time and needs to be queried at run-time. It is then the task of the developer to retrieve the location of the functions he/she needs and store them in function pointers for later use. This is made easier through the use of GLEW which performs the taskof retrieving locations and storing them in pointers without user intervention. It is hence essential to link these libraries to the application before commencing development [13].

The first step in designing any graphical application is the creation of a window as shown in Figure 4.1.1(a), of sufficient resolution and dimensions. This window specifies an area on the screen of a computer wherein 2D and 3D rendered graphics are to be displayed. Furthermore, users can view the outcome of their interactions with created graphical models and objects by means of the window.

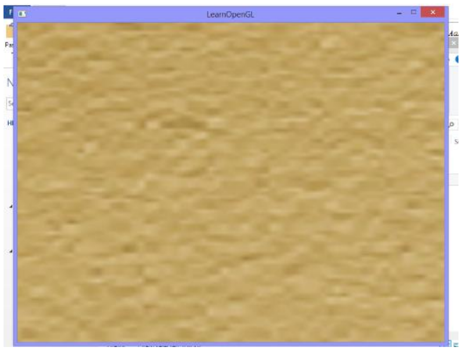

Figure 4.1.1 (a) – A Window of 800x600 resolution.

The second step involvesshader definition.In OpenGL everything is in 3D space, but the screen and window are a 2D array of pixels so a large part of OpenGL's work is about transforming all 3D coordinates to 2D pixels that fit on your screen. The process of transforming 3D coordinates to 2D coordinates is managed by the graphics pipeline of OpenGL.The graphics pipeline (Shown in Figure 4.1.1(b)) takes as input a set of 3D coordinates and transforms these to colored 2D pixels on your screen. The graphics pipeline can be divided into several steps where each step requires the output of the previous step as its input. All of these steps are highly specialized (they have one specific function) and can easily be executed in parallel. Because of their parallel nature most graphics cards of today have thousands of small processing cores to quickly process your data within the graphics pipeline by running small programs on the GPU for each step of the pipeline. These small programs are called shaders.

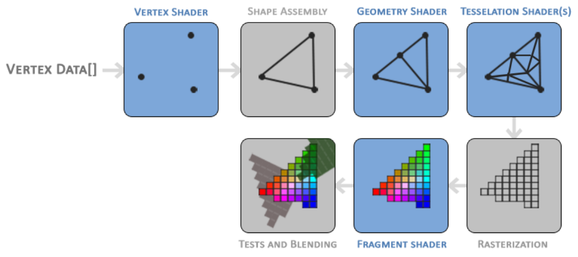

Figure 4.1.1(b)– The Graphics Pipeline

Some of these shaders are configurable and it is these shaders that the application should be concerned with – more specifically the Vertex and Fragment Shaders. The first part of the pipeline is the vertex shader that takes as input a single vertex. The main purpose of the vertex shader is to transform 3D coordinates into different 3D coordinates (more on that later) and the vertex shader allows us to do some basic processing on the vertex attributes.Hence, all transformations that need to be performed on the room model will be done here. The main purpose of the fragment shader is to calculate the final color of a pixel and this is usually the stage where all the advanced OpenGL effects occur. Usually the fragment shader contains data about the 3D scene that it can use to calculate the final pixel color (like lights, shadows, color of the light and so on)[14]. Hence, the fragment shader will be responsible for handling all texture related information of the room model.

4.1.2 Loading the Room Model's .obj and .mtl files

The third step involves importing the room model obtained from previous stages in Wavefront.obj, into the application. This is facilitated by the use of model importing libraries such as Assimp(OpenGL Asset Import Library). Assimp is able to import dozens of different model file formats (and export to some as well) by loading all the model's data into Assimp's generalized data structures. As soon as Assimp has loaded the model, we can retrieve all the data we need from Assimp's data structures. Because the data structure of Assimp stays the same, regardless of the type of file format we imported, it abstracts us from all the different file formats out there.[11]The Assimp libraries are hence configured to process the imported .obj file and its.mtl file that describes the room model and it's textures. In order for OpenGL to be able to process and render textures however requires a library such as SOIL. SOIL stands for Simple OpenGL Image Library and supports the most popular image formats

4.1.3Setting up a First Person Camera

Finally, a first person camera is configured to navigate within the loaded room model. In OpenGL, the illusion of a first person camera is created by moving the scene dynamically as per user input, giving the impression of constant free-flow movement. This is made possible by dynamically transforming the co-ordinates of the loaded room model by multiplying them with Model, View and Projection matrices. A model matrix transforms coordinates from local object coordinate space to World Space (The coordinate space of our application). The View Matrix is responsible for transforming world space coordinates as per the 'camera's view space. Finally, the Projection matrix transforms the coordinates into Clip Space, which basically clips objects that lie outside the screen. Each of these transformations need to be performed on each coordinate that enters the Vertex Shader. To allow for dynamicity, each of these matrices are computed at runtime based on user input and camera position.

The application is programmed to accept user input via the arrow keys and mouse through the use of specific callback functions. These functions determine the camera's changing positions which in turn affect the transformation matrices applied to the room model's coordinates. At the end of this stage, the user should be able to see the complete rendered model (Figure 4.1.3) within the application window and also navigate within it using designated keys and the mouse. User movement however is confined to the interior of the model.

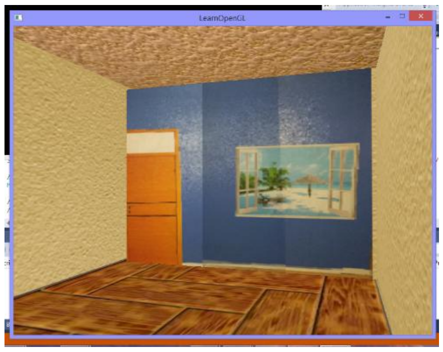

Figure 4.1.3 – Navigating the Rendered Model

4.1.4 Populating the scene with objects

Should the user wish to populate the scene with objects such as furniture, each of these would have to be designed on Blender by the designer of the application and exported in Wavefront .obj file format, as done for the room model. However, each mesh within this complex scene will have to be processed separately by Assimp before they can be rendered. The transformations that result from the movement of the camera would also have to take into account the existence of other mesh objects. Therefore, the application should be designed with special regard for scalability allowing the user to import a pre-created set of object models into the scene.

5. ROLE OF THE END USER

The final build of the application thus enables the user to create a three dimensional, virtual representation of an interior scene by carrying out the following steps-

1. Procuring the images of each of the faces of the room, two for each face.
2. Stitching the images using the Image Stitching Algorithms provided.
3. Placing the stitched images of each wall on the corresponding faces of the 'T' that corresponds to the 3D model of the room
4. Building the OpenGL application that loads and processes the model, to allow for simulation using a suitable compiler.

6. CONCLUSION

It is safe to assume that the visual accuracy of the rendered room depends solely on the quality of the images obtained in the first stage. It is preferable to mount the digital camera on a tripod, to attain minimum shift in angle while capturing each of the halves of the faces. Moreover, the two images of a single wall should have as many common visual features as possible. This in turn increases the number of inlier matches obtained as a result of the RANSAC algorithm. Images with a very low number of common features lead to fewer inlier matches although the number of key-points detected might be significantly high. It is observed that a minimum of 50% of the total number of matches detected by SIFT should lead to successful inlier matches to attain higher resolution, stitched images of significant accuracy. The accuracy pertaining to the relative placement of the stitched images with respect to the faces of the cube model however can only be judged first hand by the human eye and is not measurable.

The concept proposed in this paper can be extended to more complex room designs. Moreover, transparency within the system can be achieved by hiding from the user the entire image stitching process and code, as well as that of OpenGL. This can be facilitated by the development of a GUI which would guide the user through the four steps of modeling mentioned in Section 5 of this paper and simply display the output of each stage.

The ease with which 3D Models are created via the proposed system, allows the cost-effective creation of 3D Models of interior scenes which can be used for low budget Virtual Reality training of law enforcement officers and the like. It can also be used by real estate brokers to provide customers with rough virtual models of home interiors. Moreover, several educational institutions can create virtual representations of their campuses to be hosted on online portals. The system however is not limited to these applications and can be used in various other scenarios that require three dimensional virtual representation of interior scenes.

7. ACKNOWLEDGMENTS

Our thanks to the authors of VLFeat, for their detailed tutorials and open source library that aided us in our implementation of SIFT and RANSAC. We are also grateful to Mr. Joey De Vriesand Mr. Arnaud Masserann, whose highly elaborate tutorials on OpenGL and constantsupport made this project possible. Equally appreciated is Mrs. Grishma Sharma, faculty member of the Computer Department at K.J Somaiya College of Engineering Mumbai, India, who mentored us along the way.

8. REFERENCES

[1] WIKIPEDIA http://en.wikipedia.org/wiki/Image_stitching

[2] WIKIPEDIA http://en.wikipedia.org/wiki/Scale-invariant_feature_transform

[3] Guerrero, Maridalia, "A Comparative Study of Three Image Matcing Algorithms: Sift, Surf, and Fast" (2011). All Graduate Theses and Dissertations. Paper 1040. http://digitalcommons.usu.edu/etd/1040

[4] Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision, 60, 91-110.

[5] Dubrofsky, Elan (2007). Homography Estimation. https://www.cs.ubc.ca/grads/resources/thesis/May09/Dubrofsky_Elan.pdf

[6] J. J. Lee and G. Y. Kim. Robust estimation of camera homography using fuzzy RANSAC. In ICCSA '07: International Conference on Computational Science and Its Applications, 2007.

[7] R. Hartley and A. Zisserman. Multiple View Geomerty in Computer Vision. Cambridge University Press, second edition, 2003.

[8] WIKIPEDIA http://en.wikipedia.org/wiki/Blender_%28software%29

[9] WIKIPEDIA http://en.wikipedia.org/wiki/UV_mapping

[10] WIKIPEDIA http://wiki.blender.org/index.php/Doc:2.6/Manual/Textures/Mapping/UV/Unwrapping

[11] De Vries, Joey "Assimp." '*learnopengl.'* http://learnopengl.com/#!Model-Loading/Assimp

[12] WIKIPEDIA http://en.wikipedia.org/wiki/Wavefront.obj_file

[13] De Vries, Joey "Creating a window." '*learnopengl.'*http://learnopengl.com/#!Getting-started/Creating-a-window

[14] De Vries, Joey "Hello Triangle." '*learnopengl.'*http://www.learnopengl.com/#!Getting-started/Hello-Triangle

[15] De Vries, Joey "Textures." *'learnopengl.'* http://www.learnopengl.com/#!Getting-started/Textures