Multiple Camera Mapping

Rodrigo Marques [1,2], Pablo Bioni [1], Bruno Feijó [2]
[1] Globo Group, Visual Effects Research Group
[2] Pontifical Catholic University of Rio de Janeiro
rodrigo.silva@tvglobo.com.br, pablo.bioni@tvglobo.com.br, bfeijo@inf.puc-rio.br

1. Abstract

Most of computer graphics scenes try to represent real world locations. The traditional modeling techniques are not suitable for telenovelas. The computer graphics artists spend a long time to recreate and, sometimes, it is not similar enough. An alternative approach is to use camera mapping techniques and build simple geometries of the reconstructed area for projection. But this process takes a long time to achieve a photorealistic result.

For small places and objects use photogrammetry technique is a good solution for modeling and rendering. But working with a huge place, with high number of pictures, the result could be useless.

This paper presents a new procedure to reconstruct and shade automatically an area captured by a sequence of photos or a high definition video (e.g,. 4K cameras), with artistic interference if needed.

2. Motivation

Typically, a virtual 3D set is based in real world location, and to produce the virtual version we need to take a very large amount of photos, videos and notes. Moreover, we need to allocate few artists to recreate the models and create the textures.

Another useful approach is use the camera mapping technique to produce the lighting and details. This methodology accelerates the process because the artist can recreate only simples geometries to project the texture. But the projection texture is not trivial to create and remove distortion artifacts.

Thus, we propose to create a engine that uses the photos and videos, creates a simple representation of the scene, calibrates the camera's position and automatically chooses which cameras are better to project in a specific virtual camera position.

The first project that used the technique was "Passione", which used the a version of Glyph Maya projection toolkit. After that we implemented a new algorithm for RealTime (OpenGL) and Optix.

1) Point Cloud of a Race Track. 2) Modeling and Camera Projection Scene

3. Project Concept

The system concept uses OpenCV to align and calibrate the cameras. After creating the camera parameters, and 3D orientation, it produces and exports the sparse point cloud.

The next step is reconstruct the scene and create a simple geometry version, based on the point cloud. With the geometry and the cameras, we develop a optimized data structure to fast select which cameras can produce a better projection to a virtual camera. The method is viewpoint dependent, thus, it is a multi-resolution method, because when the camera is close to an object, it will select the best camera to project. Also, we create a robust technique for texture combination.

4. Camera Calibration and Point Cloud Generation

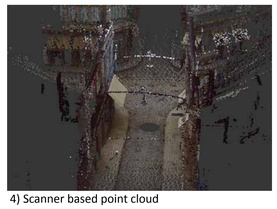
The camera calibration uses a feature based technique with SIFT and RANSAC to estimate the cameras poses. This procedure is well performed by OpenCV.

Using the Stereo Matching algorithm, we generate the point cloud representation of the scene, this procedure is done in GPU through the OpenCV GPU support. The Point Cloud is filtered to remove noise and outliers (that could be a cluster of points) and saved.

We also support data from scanner using PLY file format to exchange between softwares, and PRT file format for final storage, due the compatibility with Thinkbox Krakatoa and the compression capacity. A typical scanner file has 450 million points and cover an area of 12500 m². A Point Cloud generated by images has between 120 to 500 thousands points and can cover 10 times the scanner area.

Therefore, despite the data type and scene representation, the point cloud from a scanner and image sequence has differences in the density. The last version of the software supports data from Agisoft Photoscan.

3) Image sequence based point cloud



4) Scanner based point cloud

5. Simple Geometry Reconstrution

We could use an algorithm to create a mesh for the Point cloud, but, normally the mesh is too complex and heavy that is hard to recreate the topology in a commercial software. Also, the texture maps are not understandable for humans, thus, it is hard to modify.

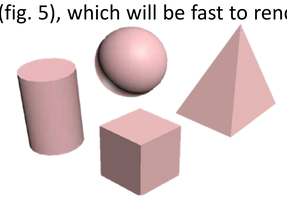
Our approach simplifies the scene using four types of parametric objects (fig. 5), which will be fast to render using an optimized raytracer.

The process starts with the bounding box computation of the point cloud. The system subdivides the scene using a plane, starting from the top to the bottom (we assume that the sky is on the higher vertical values). At each step, the algorithm tries to align the parametric objects to cover the points. If the alignment is not good enough, the algorithm subdivides.

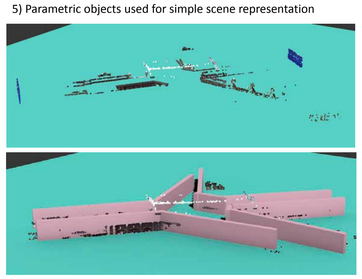
Each step tries to combine the previous objects with the new point set, belonging to the space between the planes of the step (i) and step (i+1). The procedure runs until the bottom is reached. Each step has an error minimization step.

After the simple reconstruction, a global error minimization and a collision solver algorithm finishes the process.

At this point, we can generate the mesh or use the parametric representation.



5) Parametric objects used for simple scene representation



6) Plane Intersection
7) First align of boxes to the points

6. Projection and Rendering

With the simple geometry and the cameras information, we can start the projection and shading. To perform this, we need the virtual camera to minimize the projection error.

For each pixel we compute the best camera for the projection, based on the projection surface, camera position and rotation. We assume that all camera has the same capture setup (ISO, aperture). The factors are:

- Non occluded area: We need to know if the camera can see the object.
- Perspective error: Using the normal, we rank the cameras by the maximum $D_{projection\ camera} \cdot D_{virtual\ camera}$(D is direction). This forces to use the projection camera with the closest orientation to the virtual camera.
- Surface perspective error: Minimizing $N_{surface} \cdot D_{projection\ camera}$, we select the most perpendicular projection camera.
- Camera Distance: We select the closest camera to the virtual camera.
- Surface Distance: We select the closest camera to the surface.
- Artistic selection: This is an artistic parameter.

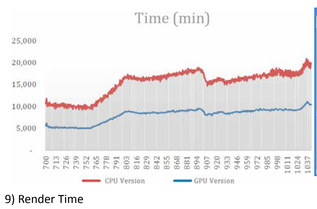
All the factors are combined by an weight vector. If a camera does not cover a area, the algorithm selects the next one.

To combine multiple cameras, we use a blending factor, that is the surface perspective error, combined using an overlay equation. Also, we compute a Monte Carlo ray casting to blend with other cameras.

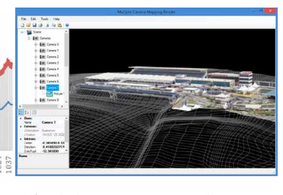
7. Results and Conclusion

The camera projection using simple models and high quality images, can produce a very realistic scene, with a very fast rendering procedure. Also is possible to change the pictures to achieve other lighting conditions (the origin and axis vectors must be transformed).

The method is under development, to achieve good results we still need an artist to improve the automatic modeling and trick the projection system. However, the total time for scene setup was significantly reduced.



9) Render Time



10) Multiple Camera Projection Application