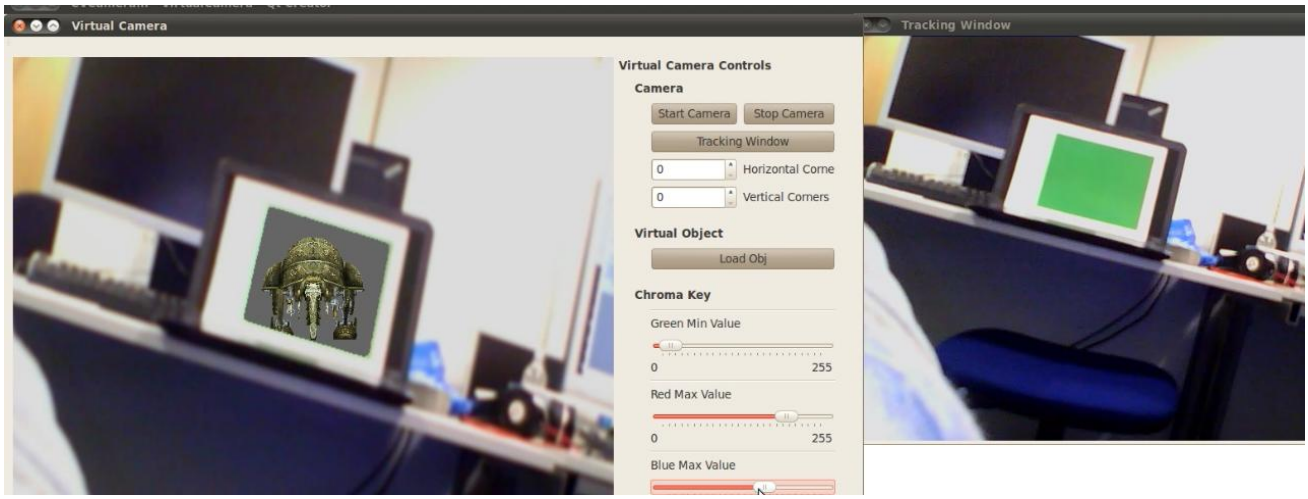

Virtual Camera

Raaj Gupte

NCCA, Bournemouth 2011-2012

1. Introduction

Movies have always employed various tricks to make the impossible seem believable within the movies. Giant Monsters and Flying Spaceships have invaded on the movie screen many times with the help of these tricks. Before the use of computers, these tricks, known as Special Effects, were achieved using models and miniatures. Various Camera tricks used to be employed to incorporate these models with live action. Since these effects were shot in camera, the results could be seen immediately on film.

As technology progressed, various computer graphics techniques started being employed to achieve the same tricks on screen, adding more realism and believability to the Effects. With the advent of Non-Linear Editing and Chroma-Keying tools on the computer, shooting on a Green Screen/Blue Screen became one of the key techniques for implementing visual effects. In today's Film, Television and Games productions, shooting on a green screen is very common. Even in productions where spectacular effects are not required, scenes are often shot on green screen to replace the background with any desired location, eliminating the need to construct physical sets, thus minimizing the cost of production.

Most of the Compositing, Chroma-Keying, Green Screen removal however is done in Post-Production. The Virtual Elements such as scenes or effects are added later using Composting Software. This process is time consuming; giving results only after post-production work is done by a Visual Effects Company. The Director may or may not be entirely satisfied with the finished product creatively and may ask for tweaks and changes in the shot, resulting in extra Visual Effects work.

This problem can be substantially resolved by using a Virtual Moviemaking Workflow. Virtual Moviemaking is defined as a process of combining Live Action Elements with Virtual Elements in Real Time. This process can be applied in many different ways such as combining Live Actors with Virtual Sets or placing CG Characters within Live Action sets.

This paper will focus on solving the problem stated above. The aim is to replace the Green Screen Background with Virtual Elements and combining them with Live Actors in Real Time. The Virtual Elements may or may not be High End Models or Textures. Using this process, Filmmakers can get a better idea of the final shot as they shoot the scene, minimizing reworks in Post-Production.

The developed application does not limit itself to having a combination of real actors and virtual environments only. It is flexible enough to accommodate other virtual workflows with a few tweaks in the system. Virtual Characters, Particle Systems or other simulations can be easily added to the system.

2. Previous Work

There is no fixed definition for the term Virtual Moviemaking or Virtual Production. As stated earlier, one of the definitions can be given as merging of various Computer Graphics technologies to create a workflow which combine Real and Virtual Elements in Real Time. These Virtual Workflows were used in various ways in Academy Award for Best Visual Effects winning movies such as Avatar and Hugo.

In Avatar (2009), Motion Capture was used to translate the performance of real actors into Computer Generated characters. For the movie, a Virtual Camera was developed which could display the Motion Captured CG characters in action while shooting with the real actors on set. Director of Avatar, James Cameron said of the Virtual Camera "The way we developed the performance capture workflow on Avatar, is we have our virtual camera, which allows me to, in real time, hold a camera -- it's really a monitor -- in my hands and point it at the actors and see them as their CG chartacters." (Billington 2008). Another tool that was developed for the movie was known as Simulcam. It simulated the whole CG environment on screen as they were shooting live action. With the Simulcam used with the Virtual Camera, the production team was able to view the CG elements and Live Action elements within the same frame in real time. James Cameron described the Simulcam as "We're taking our virtual production toolset and superimposing it on physical production. We turned the set on the soundstage into a capture volume and turned the physical camera into a capture virtual camera, so we were able to integrate CG characters and environments into our live action." (Billington 2008).

A similar Virtual Camera system was used for Hugo (2011). Various tracking devices were placed on the movable parts of the camera to capture its physical movements. These movements were translated into real time CG elements in place of the Green Screen (Robertson 2012). This allowed director Martin Scorsese to frame the shots with real actors placed on virtual set in real time.

Autodesk's MotionBuilder was used for real time playback of the Live Action elements combined with CG Elements for both the movies.

One of the backbones of the Virtual Camera is the use of Green Screen Chroma Keying. Although Green Screen Chroma Keying is done extensively for visual effects, most of it is done in post-production. There is very little literature or applications regarding Real Time Chroma Keying.

3. Design

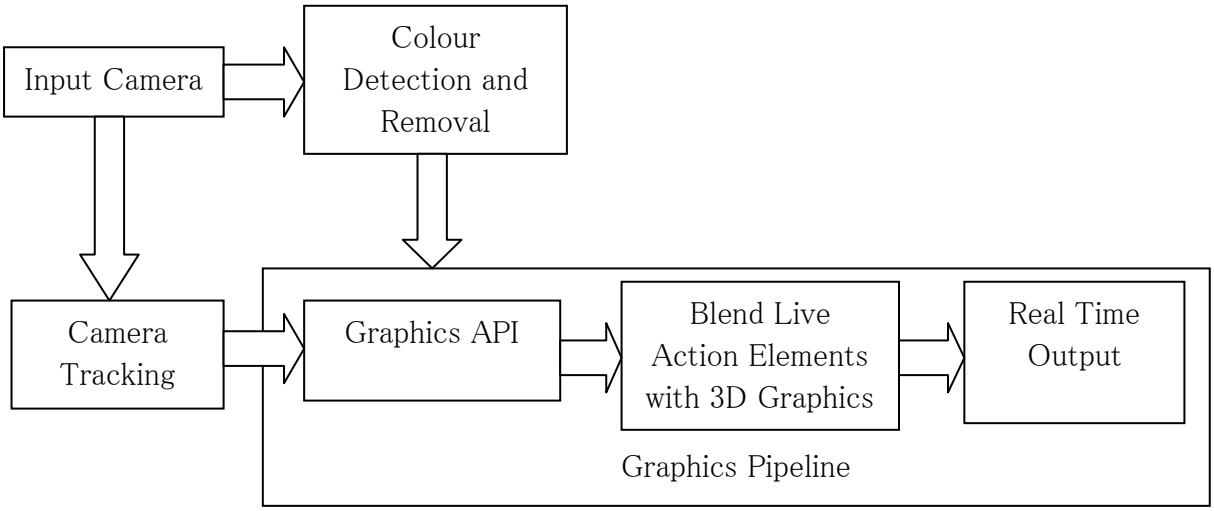

Figure 1. Diagram showing Architecture of the Virtual Camera

As seen in the Design diagram, there is a Camera which acts as the Input for the whole system. The Camera streams in live images and feeds into two Modules: Colour Detection and Removal and Camera Tracking.

Colour Detection consists of processing each input image and detecting a particular colour to be identified as background. In our case, the colour is Green. Once this colour is detected, further image processing is carried out and the background is removed or made transparent.

The output of the Camera is also fed to the Camera Tracking module. This module utilises tracking markers to detect the movements of the camera in physical space and converts it co-ordinates in the virtual space.

These movements are fed to a Graphics API which applies these transformations to the Virtual 3D elements. The virtual elements are also drawn by the Graphics API.

The Graphics API further takes in as an input the processed images from the camera and combines it with the 3D elements drawn by it.

These combined images are displayed in real time on a display by the Graphics API. The last three modules can be combined to form a Graphics Pipeline.

The camera images and the Virtual 3D elements both move in relation with the camera movements, giving an illusion that the Virtual objects are actually a part of the live action being shot.

3.1 Requirements

The Virtual Camera has been developed on a Linux machine and has some Software and Hardware dependencies to be fully functional.

3.1.1 Software

C++ is used as the main coding language with Qt SDK for developing the project. Since a combination of image capturing and processing with a graphics API for drawing CG elements is required, OpenCV is used in tandem with OpenGL.

3.1.1.1 OpenCV

OpenCV is an Open Source Library built for implementation of Computer Vision modules (Bradski & Kaehlerr 2008). It is built using C and C++ and runs on a host of Operating Systems. OpenCV is particularly suited for Real Time Applications as it utilises fast and effective algorithms for the same. OpenCV was chosen as the main Library for Computer Vision and Image Processing as it satisfies the requirements of being a fast, free open source utility. OpenCV 2.0 is the version used for this project.

3.1.1.2 OpenGL

OpenGL is a graphics API which is used for drawing 3D graphics in Computer Applications. In our project, OpenGL is used for drawing the 3D Virtual Elements. These can be Background environments or CG characters. OpenGL gives us the functionality to integrate OpenCV images into the Graphics pipeline as texture maps. Using OpenGL functions, one can further process the image streams from the input camera and combine the live action elements with 3D graphics elements.

3.1.2 Hardware

The Virtual Camera is designed to be a standalone piece of software. For its proper functioning, the Virtual Camera is however dependent on a number of Hardware components.

3.1.2.1 Digital Camera

Since the Virtual Camera takes input video streams, the most essential piece of Hardware required for its functioning is a Physical Camera which can capture video. There are no limitations on the type of the Camera that can be used except for the fact that the Camera should be able to capture images digitally and that it can be interfaced with a Computer. In order to develop a robust piece of software and make the quality of the output independent of the quality of the input camera used, a

simple low end Web Camera was used for development of this Project. If the software is to be used for production quality applications, it would be better to use a high quality Digital Camera. The Virtual Camera may function without the use of other components listed below, but without a Physical Camera, the application would not start at all. Hence a Digital Camera is the main Hardware requirement for this project.

3.1.2.2 Green Screen

Chroma Key Compositing is a predominant technique used in the VFX Industry to replace backgrounds in Live Action Productions. Either a Green Screen or Blue Screen Background is used for Chroma Keying. While both the colours have its own advantages and disadvantages for Chroma Keying, this project utilses a Green Background. The reason for doing so is that most Digital Cameras provide an undistorted Green Channel than the Red and Blue Channels (Weigert 2010). Hence it is easier to get better results using the Green Channel. Also, most VFX productions today primarily use a Green Screen for Compositing. A Green Screen Background is hence required to place the Virtual Set in its place. However, the software can be customised to use a Blue Screen or any other desired colour background.

3.1.2.3 Tracking Markers

The Virtual Camera is designed to closely follow the movements of the Physical Camera. In order to do so, a Tracking Marker is required to determine the External Parameters of the Physical Camera with respect to the World Space Co-ordinates. Using a Marker, the Position and Rotation of the Camera can be easily determined. The Marker should be a simple object which is easy to detect (Bradski & Kaehlerr 2008). A Chessboard has a regular Black and White pattern which is easily identifiable be Computer Vision Algorithms. OpenCV has a number of functions to detect a Chessboard pattern and calibrate the Intrinsic and Extrinsic Parameters of the Camera using the Chessboard pattern. Hence, a Chessboard Pattern is used as Tracking Marker for determining the Parameters of the Physical Camera. It is also very easy to create and use a simple Chessboard Pattern printed on plain paper.

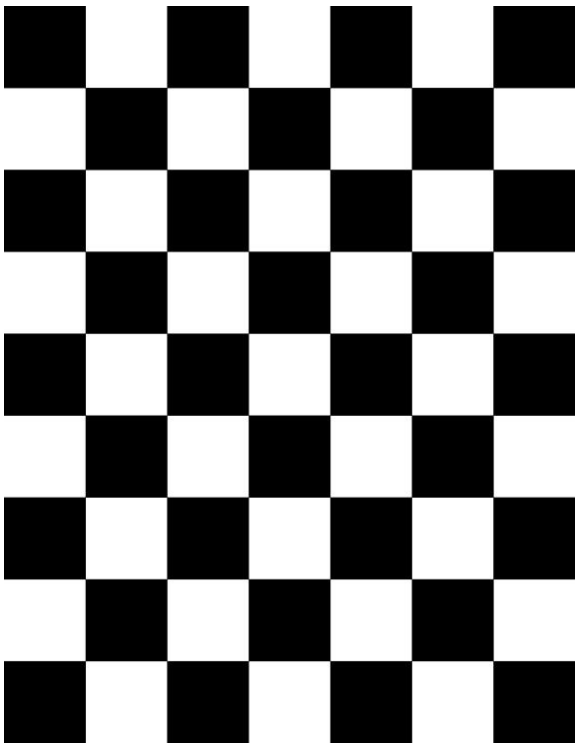

Figure 2: Sample Chessboard Pattern used for tracking. (Kuntz 2009)

4. Implementation

The aim of this project was to design and create a Virtual Camera, which takes in images from a live camera feed with Live Actors placed on a Green Screen Background and process the images to replace the Background with Virtual Elements in Real Time. The Virtual Camera would also closely follow the Physical Camera. The Physical movements of the Camera are tracked and would be emulated by the Virtual Camera on screen. Whenever the Camera Tracks, Pans or Zooms, similar movements are carried on screen and the Virtual Set along with the Live Actors would move in accordance with the camera movements. The user can film a Green Screen shoot using normal Physical Camera movements and the software would translate them on screen in real time.

As seen in the Previous Work section, such type of Virtual Production Workflows use state of the art equipments and are very costly. The goal of this Project was to create a Virtual Camera using minimum equipment to create a low cost solution. If using simple, low cost equipments in both Hardware and Software, the project is able to deliver a satisfactory output then one can build on the software using better quality equipment to make it production ready.

As seen in the Design diagram earlier, the Virtual Camera can be divided into three distinct modules:

- Colour Detection and Chroma Keying
- Camera Tracking
- Computer Graphics Pipeline

Each module is loosely coupled with other modules. Every module is a separate entity and can function individually on its own. Chroma Keying can work perfectly well with a static camera without the tracking module. Similarly, with just the Camera Tracking and the Graphics API, the Camera can be used as a joystick to move the CG Environment. All the modules will be discussed in detail further.

4.1 Colour Detection and Chroma Keying

Background detection is one of the forefront problems of Computer Vision. Background detection is the process of identifying and distinguishing elements which are static and which are moving from an image. In any application, it is harder to detect a non uniform background than a stable uniform single colour background. However, detection of a single colour background has its own set of

problems, which will be elaborated later.

For our project, Background Detection is required to identify which elements from the Live Action scene should be present within the Composited Virtual Environment and which elements should be removed. Usually, the elements to be removed will be a Green Screen Background. This can be achieved by Chroma Keying the Green colour and retaining other colours.

The process for doing this is to scan through every pixel and check for its colour value. If the colour value of the pixel is green, replace it else retain the colour value. In order to check the colour value, one needs to investigate into the RGB colour model that is used in Computers and Digital Cameras. The images produced by Digital Cameras are handled by OpenCV as RGB colour model and hence it is useful for the project. One minor difference that needs to be noticed is in the way OpenCV handles this colour model is that it uses BGR in place of RGB and hence first it is needed to convert the BGR image into RGB to handle it properly (Bradski & Kaehlerr 2008).

4.1.1 RGB Model

RGB is an additive colour model that is primarily used in Computer Graphics. It uses the Red, Green and Blue colours as primary colours for additive mixing. Other colours can be derived by adding two or all three of the primary colours.

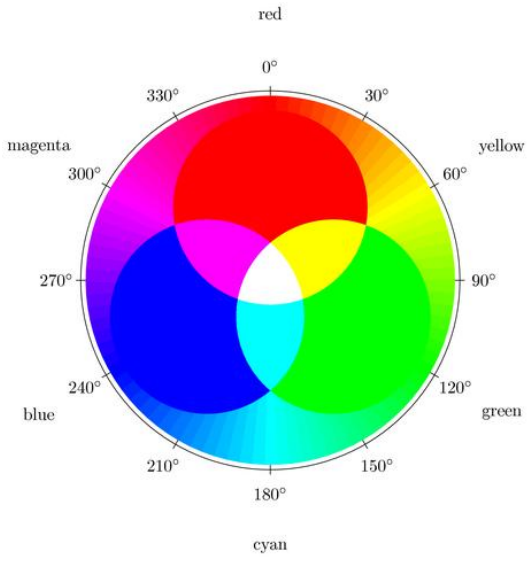

Figure 3. The circle shows the additive spectrum of the RGB colour model. The Percentage of colours added determines the colour that is produced.

In Computer Displays, colour is depicted by the colour value of a pixel. Every pixel contains three channels of data representing the Red, Green and Blue Channel from the RGB model. The percentage of each colour present in the channel determines the colour value of the channel.

The numerical representation of each channel can be done in many ways:

- From 0 to 1 with floating point numbers in between.
- From 0 to 255
- In terms of Percentage from 0% to 100%

Each channel of a pixel is defined as an 8 bit integer with values representing 0 to 255. 0 indicates least and 255 indicates the highest saturation of the colour. The channels are packed of as an array of 8 bit integers as [R,G,B]. Thus the brightest saturated Green can be represented as [0,255,0]. In order to detect any of the primary colours, it may be intuitive to detect high values in that particular channel. Bergh and Lalioti (1999) give a Principal Algorithm to get Blue Colour pixels as those pixels that have values of Blue that are higher than both Red and Green. This algorithm can be modified to get the Green Colour.

$$G>R$$

$$G>B$$

However, since an additive colour scheme is used there could be a situation where

$$G = x, R = x -1 \text{ and } B = x - 1$$

This combination would give a shade of Gray. However since it has a high value of Green which is greater than Red and Blue, it will be still detected as Green. High values of Green are present in other colours as well, such as Yellow and Cyan. In order to counter this, Bergh and Lalioti (1999) propose to add a distance constraint.

$$d=2*G-R-B$$

Hence, a minimum threshold value for Green colour and a maximum threshold value for Red and Blue colours each is added. For each pixel, these threshold values are used to check if value of the Green channel is not lower than the minimum threshold and that of Red and Blue channels do not exceed the maximum threshold.

$$G > gMin$$

$$R < rMax$$

$$B < bMax$$

Where gMin, rMax and bMax are the threshold values for Green, Red and Blue respectively.

These values can be set and changed by the user as required depending on the scene. This feature is useful for detecting and keying out different shades of Green. Also, if the Live Action elements contain a colour having high Red or Blue and a high Green value, rMax or bMax can be set to a higher value so as to avoid the colour being falsely detected as Green and being keyed out.

Having used the above algorithm to detect a wide range of Green colour, proper detection of the Green Screen is dependent on a number of external parameters like Lighting Conditions or the type of the Green Screen material used.

4.1.2 Lighting Conditions

Lighting Conditions is one of the most important parameters to be taken into consideration for a professional film or video shoot, with or without a Green Screen. The way a subject is lit affects the perceived colour of the subject. Luminance is given by the weighted sum of the three channels Red, Green and Blue (Poynton 2003). Luminance denoted by Y is given as:

$$Y=0.2126R+0.7152G+0.0722B$$

Thus, if an object that is not Green in colour is lit too much, the value of the Green channel would also be high, incorrectly detecting the object as Green. Consequently, if a Green coloured object is dimly lit, the Green channel value may be too low and hence it may not be detected as Green. Hence it is necessary to properly lit the scene, where object colours are detected as their natural colours and not be influenced by too much bright or dim light falling on them.

4.1.3 Green Screen Material

The object/material used as a Green Screen should not be highly reflective and should preferably be uniformly plain without any creases or folds (Foster 2010). This avoids creating hotspots or non-detection of the creased area of the Green Screen.

For testing purposes, in this project an A4 size glossy paper printed with Green Colour having RGB value of [0,255,0] has been used. The paper reflects light and can be easily bent or folded. This is done in order to test worst case scenarios.

4.2 Computer Graphics API

OpenGL is used in this project as the main Graphics API. The use of OpenGL in this project is twofold:

• Image Processing

• Drawing of 3D Virtual Objects

4.2.1 Image Processing

Although OpenCV is used for Image Processing in this project, OpenGL is used in tandem with OpenCV to enhance the Image Processing Capabilities of the software. OpenGL provides excellent functionality with regards to Texture Mapping and Alpha Bending.

4.2.1.1 Texture Mapping

In order to use OpenCV in tandem with OpenGL, the OpenCV input image from the camera is used as an OpenGL Texture. The input image can be pasted on an OpenGL quad surface as a 2D Texture Map. This texture will get updated as and when the input image gets updated as per the frame rate. The following commands for texture mapping are used:

glBindTexture

This is command indicates the texture id that will be applied to the surface.

glTexImage2D

This is the command that actually maps the texture which can be an image file to the object. In this case, one has to map an OpenCV image, which is stored as a Matrix data. This is the main step required for interfacing the input images from the webcam with OpenGL. Once this is done, the textured plane can be drawn into the OpenGL framebuffer.

4.2.1.2 Alpha Blending

OpenGL provides very good functionality for Alpha Blending. Since it is required to combine Live Action images from the camera with Virtual Sets, Alpha Blending is very useful in this project. The principle idea is to have Alpha value set to transparent for those pixels which contain Green and are to be replaced. Thus with the Green Screen pixels completely transparent, one can easily see the virtual set placed directly behind the transparent pixels.

For Alpha Blending, the following OpenGL functions are used:

glEnable (GL_BLEND)

glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)

Documentation from www.opengl.org states that the first function enables Alpha Blending, thus blending the values in the colour buffer with the fragment colour values

Using glBlendFunc the RGBA pixel value is calculated as:
Rd = min(kR, Rs sR + Rd dR) Gd = min(kG, Gs sG + Gd dG) Bd = min (kB, Bs sB + Bd dB) Ad = min (kA, As sA + Ad dA)

With GL_SRC_ALPHA as the sfactor and GL_ONE_MINUS_SRC_ALPHA as dfactor, the above equation is reduced to

Rd = Rs Gd = Gs Bd = Bs Ad = As

This makes the pixel values fully transparent, rendering the colour values from the framebuffer on to the screen. Hence the virtual set is first drawn into the frame buffer and then draw the live camera image plane with Alpha value = 0 for pixels which need to appear transparent.

4.2.2 Drawing 3D Graphics

OpenGL is primarily used as a Graphics API for drawing 3D Graphics. 3D models stored in the obj file format are used as virtual objects within the project. Using OpenGL commands, the obj file is loaded into the scene. One can load in a texture image file and apply it to the object using texture mapping as discussed above.

4.3 Camera Tracking

The main difference between the Virtual Camera and a Real Time Chroma Key Application such as one used for Weather Maps is the use of Camera Tracking. The Virtual Camera is designed to have an immersive environment within the Virtual Set. As the Camera moves around in Physical Space Co-ordinates, the Virtual set should move in accordance with the motion of the Physical Camera. For doing this, one needs to map the Camera position and orientation within World Space co-ordinates to the OpenGL World co-ordinates. This can be done by using tracking markers and finding the intrinsic and extrinsic parameters of the camera.

4.3.1 Camera Model

The Camera model in OpenCV is based on a simple pinhole camera. Given the focal length f, the image of an object x on an imaging plane is given by (Bradski and Kaehler 2008):

$$x = f\, X/Z$$

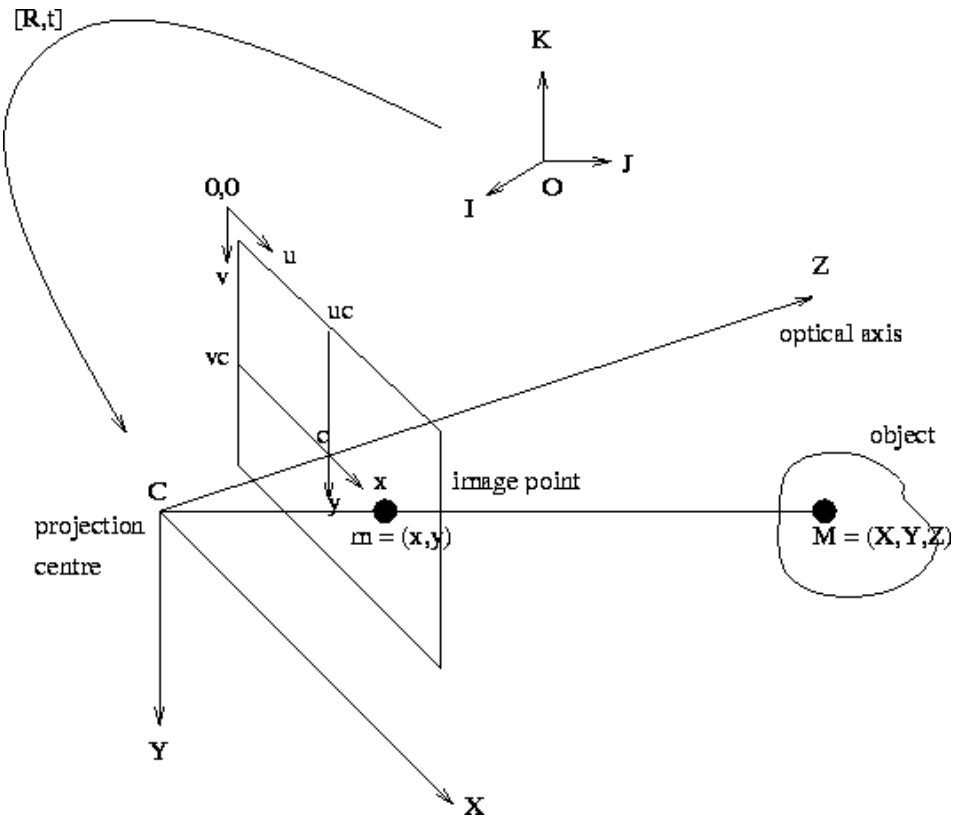

Figure 4: Figure shows the co-ordinate system for Camera Caliberation (Owens 1997)

A view is obtained by projecting 3D points on an image plane multiplied by a Perspective Transformation.

$$s\, m' = A[R|t]M'$$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

(X,Y,Z) are the world space co-ordinates of a 3D point. (u,v) give the pixel co-ordinates of the projection point. A is the camera matrix having the focal lengths (fx,fy) and intrinsic parameters (cx,cy). R and t give the Rotational Matrix and Translational Vectors.

The focal length remaining same, the intrinsic parameters of the camera do not change and can be calculated once. For our project, these are calculated and stored as intrinsics.xml file. These can be later used for getting the extrinsic parameters rotation R and translation t which can be computed as:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

$$x' = x/z$$
$$y' = y/z$$
$$u = f_x * x' + c_x$$
$$v = f_y * y' + c_y$$

The cvFindExtrinsicCameraParams2() function is used to get the external parameters of the camera. Thus using a few 3D points, knowing their projections and the camera intrinsic parameters, one can get the Rotational and Translational data of the camera for that frame. One can get the 3D points and their projections using Tracking Markers such as a chessboard pattern.

4.3.2 Tracking Markers

OpenCV has some robust functions to find image patterns. A chessboard is a very simple image pattern yet very useful for detection (Bradski and Kaehler 2008). A chessboard pattern has a number of corners which can be detected using matrix of second order derivatives of the image intensities (Harris 1988).

$$R = \det M - k(trace\, M)^2$$

$$\det M = \lambda_1 \lambda_2$$

$$trace\, M = \lambda_1 + \lambda_2$$

Shi and Tomasi (1994) calculate the score for corner detection using eigenvalues as:

$$R = min(\lambda_1, \lambda_2)$$

R can be detected as a corner if it is greater than the eigenvalues (Sinha 2010)

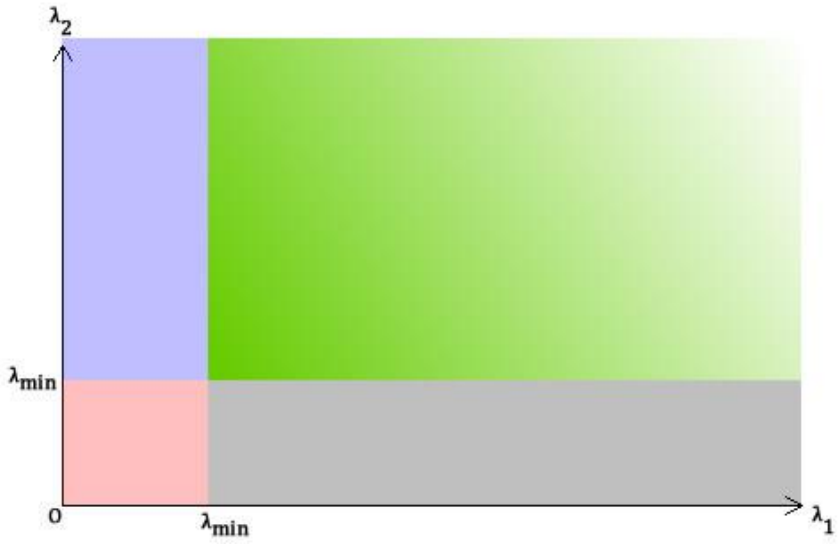

Figure 5: Figure shows detection of corners. Green is the region of pixels as corners (Sinha 2010)

OpenCV uses the Shi Tomasi Algorithm for corner detection. The findChessboardCorners() function is used for detecting a chessboard pattern. Once a chessboard pattern is found, the drawChessboardCorners() function is used to mark these internal chessboard corners.
Once the corners are detected, one can find the Extrinsic parameters of the camera for each frame, thus making Camera Tracking possible.
These parameters are then converted into OpenGL co-ordinates and the virtual objects can be moved in accordance to the camera movements.

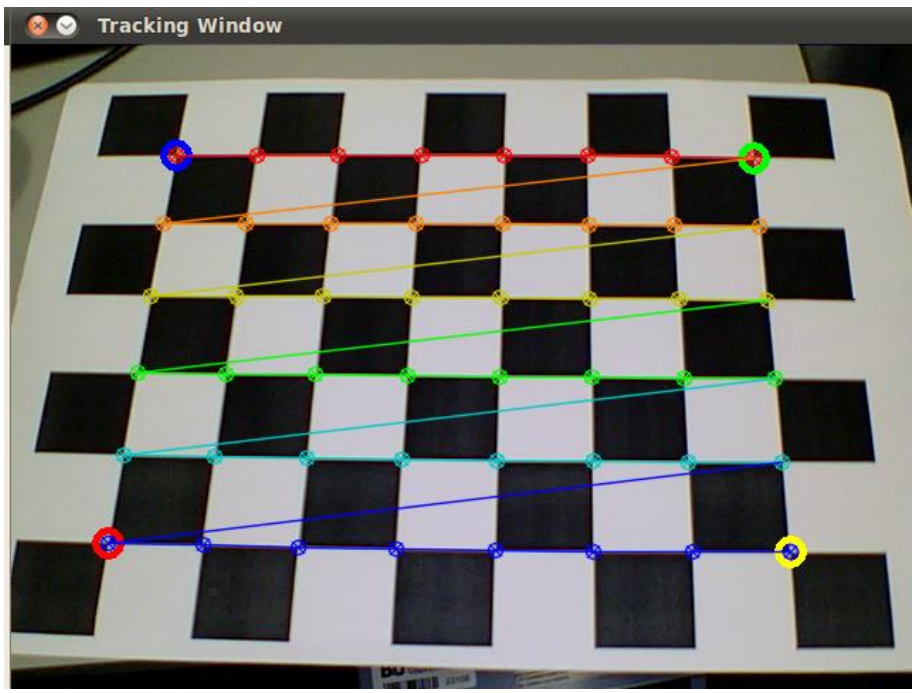

Figure 6: Figure shows the chessboard corner detection by the program.

5. Results

The Virtual Camera was designed to provide a tool which could blend in both live action and CG elements in real time. The project was developed by merging technologies like Computer Vision and Real Time Graphics processing together. Since this was an image processing based project and the output was to be seen on the screen as a composite scene, visual accuracy was of importance. This was based on two parameters: How efficiently does the software detect the Green colour and clears it out and how well the Camera is tracked.

5.1 Chroma Key Efficiency

As stated earlier, a sheet of A4 paper on which green colour was printed was used to simulate a Green Screen. Since A4 is a relatively small size, the paper could not act as a full-fledged green screen for Background Replacement. Due to this small size, the full background could not be replaced. However, green colour was convincingly detected within the area of the small sheet of paper.

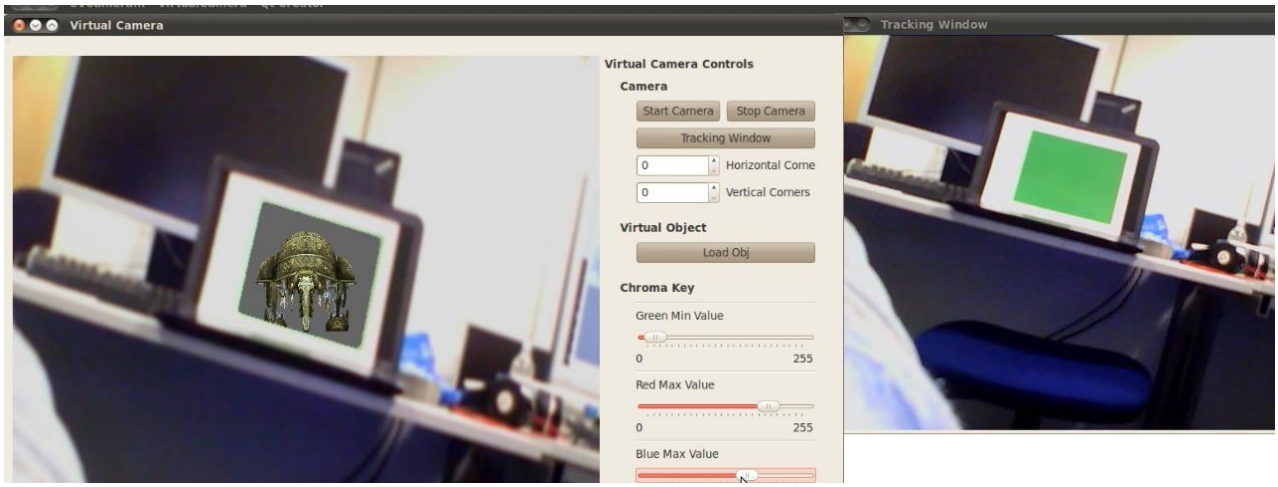

Figure 7. Image shows the removal of green screen with a 3D object.

When passing an object (hand) between the camera and the green screen, green pixels around the edge of the hand were detected. However, two different objects,the hand and the green screen were clearly detected and differentiated. Also the virtual object could be clearly seen between two fingers, thus making the test successful.

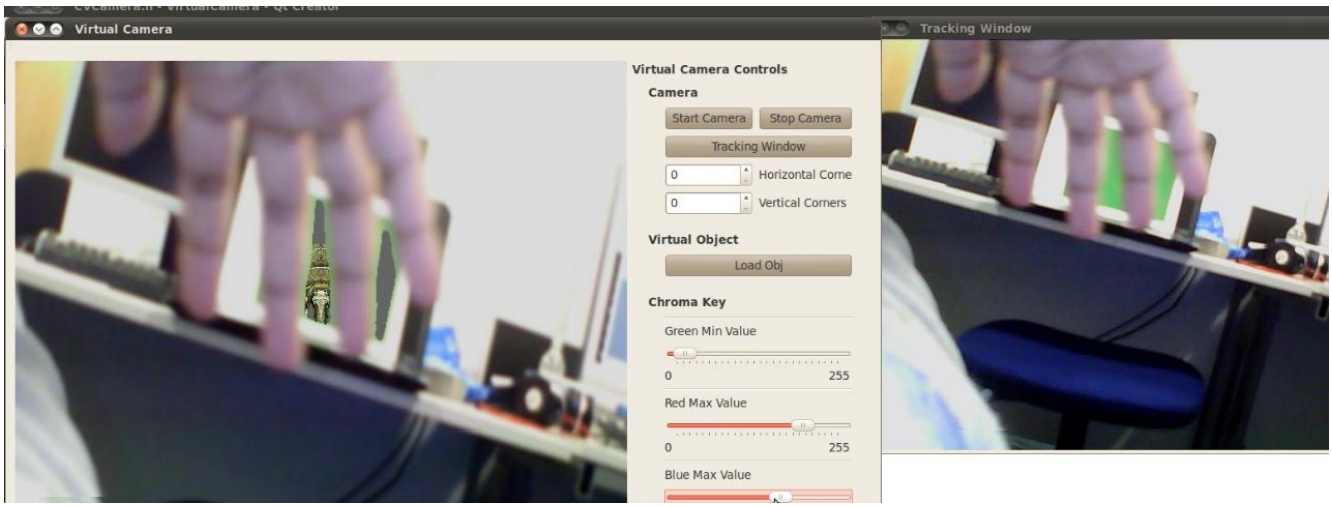

Figure 8. Green pixels around edges of the hand and virtual object seen between fingers can be seen.

Whenever the camera was moved suddenly, keeping the green screen paper in view, the detection of green colour was lost. This resulted because of non-uniform reflectivity of the paper and sudden jerky movements of the webcam. However, with adjustment in the threshold values of R, G and B, this was rectified.

Given the quality of the web camera and the green screen paper being used, the results for Green Screen Detection were fairly satisfactory. With a better quality Digital Camera and a regular standard Green Screen, these errors can be rectified.

5.2 Camera Tracking Efficiency

The efficiency of the Camera Tracking Algorithm can be determined by how closely the Virtual Object moves with respect to the camera movements. Mapping the camera co-ordinates of OpenCV with those of OpenGL was tricky. The translational parameters were scaled down whereas the rotational parameters were scaled up to match the translations and rotations in OpenGL.

The Virtual Object translated and rotated properly with respect to camera movements. However, due to jerky movements of the handheld web camera, momentarily loss in the detection of the chessboard tracking markers. This resulted in jerky movements of the virtual object, with the object jumping long distances within a second.

When the camera was steadily moved, the object moved with varying degrees of accuracy.

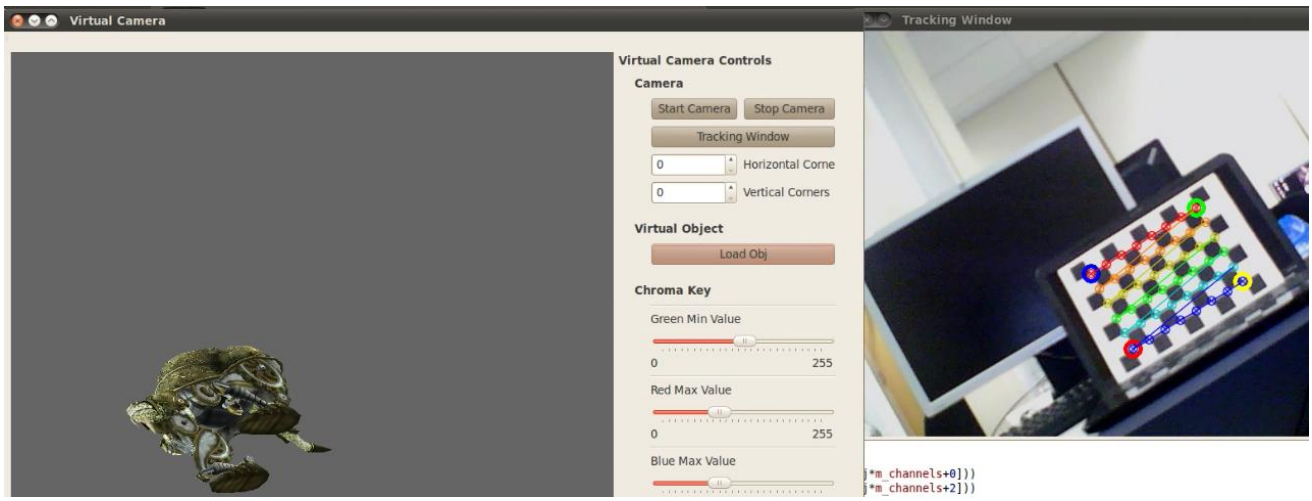

Figure 9. Rotation of the Virtual Object with respect to tracking marker and camera movement.

5.3 Combined Results.

Due to the sizes of the green screen paper and that of the chessboard tracking markers, it was difficult to move the virtual object with respect to the camera while keeping the green screen within the frame. However, when the marker and the green screen were perfectly within the frame, the object moved and rotated according to the camera movements.

Thus using a simple web camera and a printed green screen the application was tested and found to give satisfactory results with the resources available.

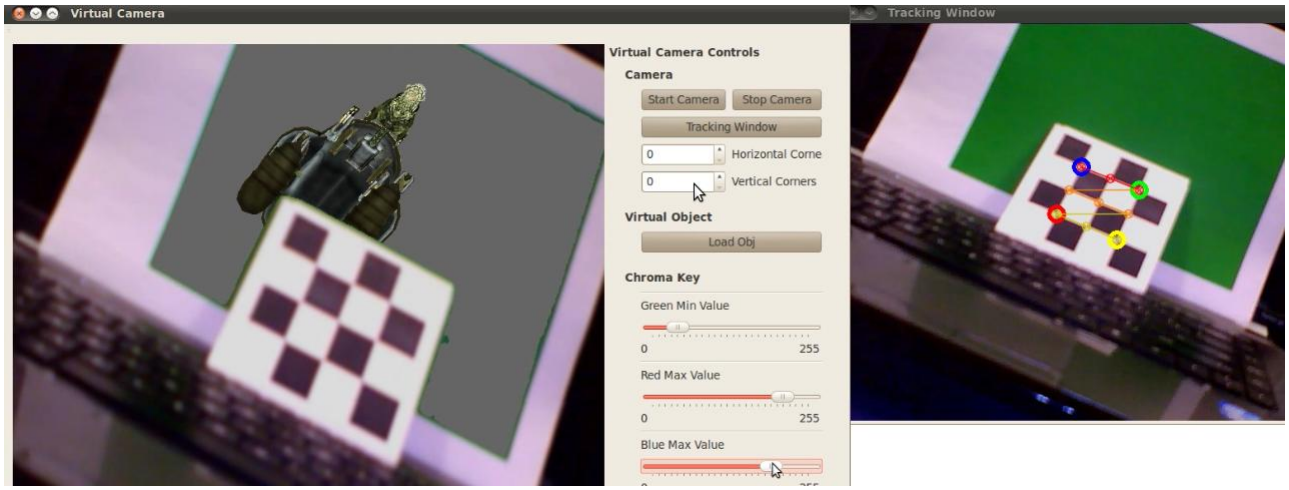

Figure10. Image shows tracking marker placed on a green screen

6. Future Works

In order to make the Virtual Camera production ready, a number of features need to be implemented. The biggest challenge is to have a stable camera tracking system. The current implementation uses a very simple web camera for input and a chessboard pattern for tracking. There are numerous problems using this approach.

The image quality of a webcam is a bare minimum and while due to its light weight, has very little stability while moving the camera around. This gives rise to jitter and shaking and has problems in detecting both the Green Screen elements as well as the tracking markers. While moving the camera around, due to the shake effect of a hand-held camera, the program loses the tracking corners and regains them, resulting in random changes in position and orientation of the Virtual Elements. The program needs to be checked with a camera with a better image quality and stability to check if the tracking errors can be minimised by this.

A tracking system with more than one tracking markers should be implemented. Currently it is not possible to use more than one marker as the program loses its tracking if more than a single chessboard pattern is detected in the scene. In order to move the virtual world, the tracking marker must be detected at all times within the frame. Using a single tracking marker limits the range of the camera movement.

An ideal implementation would be to have tracking markers all over the room giving 360 degree coverage of the position and orientation of the camera in the world space. This would give one freedom to move the camera in any direction or position and the virtual world would move according to the movements of the camera without any loss in tracking.

Another implementation could be using a Gyroscope and Accelerator attached to the camera to feed the physical position and orientation of the camera into the program without having to use tracking markers. This would free the camera from the constraints of shooting in a studio with tracking markers and scenes can be shot outdoors with just Green Screens placed in positions wherever Virtual replacement is required.

In order to get depth information about the scene leading to better camera tracking, using a pair of stereoscopic cameras were tried. This led to bandwidth issues on USB ports, giving fatal errors. Hence only a single camera was used. The usage of stereoscopic cameras can be further explored to

turn the application into a 3D visualisation tool. This would make the application useful for 3D previsualisation and shooting live scenes in 3D.

The plugin can be developed to hook the output of the Virtual Camera to Autodesk's MotionBuilder. MotionBuilder is used in Virtual Production workflows for real time capture and display of motion controlled digital characters.

The Application can be easily converted into a Virtual Reality, Augmented Reality or Mixed Reality Application by changing some functionality of its internal modules. Thus the application can be used in various industries such as medical imaging, flight simulation and defence which make use of Virtual Technologies.

The software can be ported onto Ipad, Iphones or Android phones. Using the internal sensors of these devices, tracking can be easily implemented on these devices.

Refrences

Avatar, 2009. Film. Directed by James Cameron. USA: Twentieth Century Fox Film Corporation.

Autodesk Whitepaper, 2009, The New Art of Virtual Moviemaking
Available from:
http://area.autodesk.com/userdata/static/3dec09/the_new_art_of_virtual_moviemaking.pdf
[Accessed 01 May 2012].

Billington, A., 2008, A Brief Look at the Technological Advancements in James Cameron's Avatar. Available from: http://www.firstshowing.net/2008/a-brief-look-at-the-technological-advancements-in-james-camerons-avatar/ [Accessed 29 Jul 2012]

Bradski, G., & Kaehlerr, A., 2008, Learning OpenCV: Computer Vision with the OpenCV Library. 1st ed. Sebastopol: O'Reilly Media.

Foster, J., 2011, Choosing the Right Green Screen Materials. *Available from:*
http://provideocoalition.com/index.php/lightscameraaction/story/choosing_the_right_green_screen_materials/*[Accessed 26 Jul 2012]*

Harris, C., & Stephens, M., 1988, A Combined Corner and Edge Detector. Available from:
http://www.bmva.org/bmvc/1988/avc-88-023.pdf [Accessed 26 Jul 2012]

Hugo, 2011. Film. Directed by James Martin Scorsese. USA: Paramount Pictures

OpenCV, 2009, Camera Calibration and 3D Reconstruction. Available from:
http://provideocoalition.com/index.php/lightscameraaction/story/choosing_the_right_green_screen_materials/[Accessed 26 Jul 2012]

OpenGL 3.3 Reference Pages. Available from: http://www.opengl.org/sdk/docs/man3/ [Accessed 26 Jul 2012]

OpenGL 3.3 Reference Pages. Available from:
http://www.opengl.org/sdk/docs/man3/xhtml/glEnable.xml [Accessed 26 Jul 2012]

Owens, R., 1997, Camera Calibration. Available from:
http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT9/node2.html
[Accessed 26 Jul 2012]

Robertson, B., 2012, Magic Man. Available from:
http://www.cgw.com/Publications/CGW/2011/Volume-34-Issue-9-Dec-Jan-2012-/Magic-Man.aspx
[Accessed 03 Aug 2012]

Sinha, U., 2010, The Shi-Tomasi Corner Detector. Available from:
http://www.aishack.in/2010/05/the-shi-tomasi-corner-detector/ [Accessed 03 Aug 2012]

Shi, J., & Tomasi, C., 1994, Good Features To Track. Available from:
http://www.ai.mit.edu/courses/6.891/handouts/shi94good.pdf [Accessed 26 Jul 2012]

Weigert, A., 2012, Bluescreen vs Greenscreen - How to choose. Available from:
ttp://www.awn.com/blogs/tracking-marc/bluescreen-vs-greenscreen-how-choose[Accessed 24 Jul

2012]