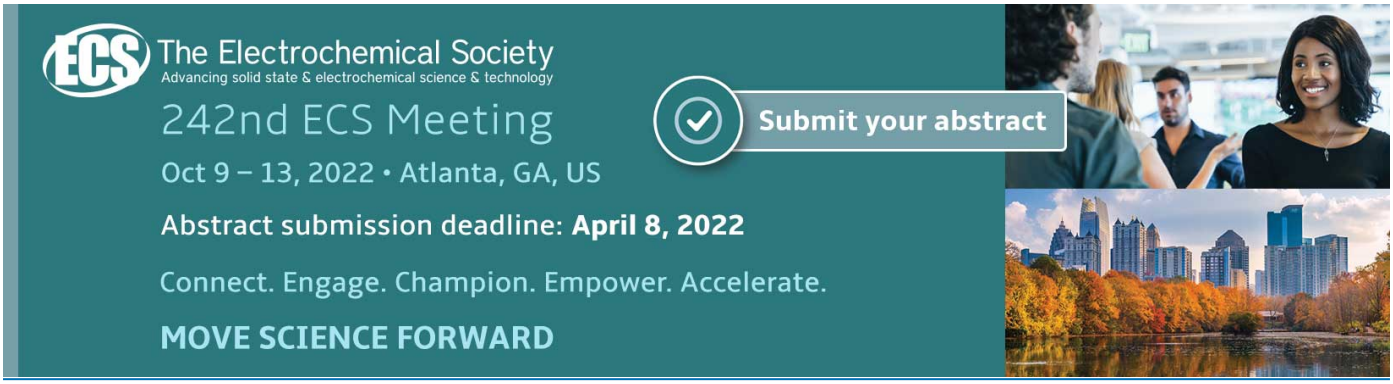
3D-modeling of Augmented Reality objects using Shi-Tomasi corner detection algorithms



3D-modeling of Augmented Reality objects using Shi-Tomasi corner detection algorithms

O M Demidenko, N A Aksionova, A V Varuyeu and A I Kucharav

F. Skorina Gomel State University 104 Sovetskaya street, Gomel 246024, Belarus

Abstract. This article covers development of the Python-based software module for Blender 3D, as well as it covers research of Shi-Tomasi corner detection algorithm using the developer's construction documents. The corners detected may be used for further three-dimensional modelling, replanting not requiring adjustment of the construction documents, or may be used for retrieval of the accurate data.

1. Introduction

Blender 3D uses "GoodFeaturesToTrack" modelling support module derived from the elaborated utilities library of BluePrint SDK to facilitate the process of 3D-modelling. The basis of the module lies in the function of corner detection of developer's layouts for further modelling of architectural designs, buildings, houses, apartments, and etc. This process provides for facilitation of development of a three-dimensional model, ensures introduction of initial adjustments in the further three-dimensional model not requiring changes in the construction documents themselves. One further peculiarity of this product is that the final layouts serve as the AR marks, which are used for implementation of the mobile application for viewing the prepared three-dimensional model.

2. Shi-Tomasi algorithm

This algorithm is used for search for and detection of view angles, which are to be in the grayscale. View angles are mainly identified as regions with large changes in the gradient intensity in all possible dimensions and directions. When analyzing a digital image, variation in pixel light intensity (x,y) is suggested to be measured by shifting the square window with the center in (x,y) by one pixel in each of eight directions (up, down, rightwards, leftwards and in four diagonal directions). The window size is often selected equal to 3x3, 5x5 or 9x9 pixels. Thus, for each of the directions (u,v), where

$$(u,v) \in \{(1,0),(1,1),(0,1),(-1,0),(-1,-1),(0,-1),(1,-1)\}.$$

The light intensity variation is calculated as follows:

$$V_{u,v}(x,y) = \sum_{\forall a,b} \left(I(x+u+a, y+v+b) - I(x+a, y+b)\right)^2 ,$$

where I(x,y) is the pixel light intensity (x,y) of the original image.

Window W for Image I (usually its size is selected equal to 5x5 pixels, but it may be selected different depending on the image size) with the center in point (x,y), as well as the shift of the window by (u,v) (figure 1) are considered below.

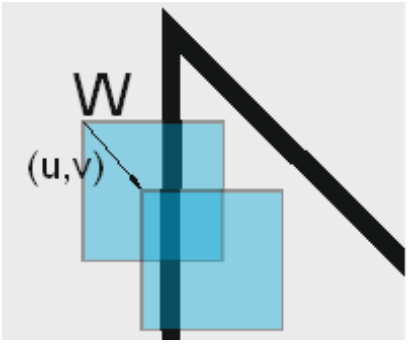


Figure 1. Example of the viewed window W and its shift (u,v).

Then the weighted total of the squared difference between window W and window W shifted by (u,v) (i.e. change in the vicinity of point (x,y) after shift by (u,v)) is:

$$E(u,v)=\sum_{(u,v)\in W} w(x,y)\left(I(x+u,y+v)-I(x,y)\right)^2 \approx \sum_{(u,v)\in W} w(x,y)\left(I_x(x,y)u-I_y(x,y)v\right)^2 ,$$

where W(x,y) is the weight function (Gaussian function or binary window is usually used).

Function E (u,v) needs to be maximized for corners detection. To determine the derivatives Taylor's theorem is applied to the above equation and, by using certain mathematical steps, the bellow is received:

$$E(u,\upsilon)\approx \begin{bmatrix} u & \upsilon \end{bmatrix} M \begin{bmatrix} u \\ \upsilon \end{bmatrix},$$

where M is the autocorrelation matrix:

$$M=\sum_{(u,v)\in W} w(u,v)\begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix},$$

where Ix and Iy are the derivatives in the directions.

The corner is characterized with large changes in function E(x,y) with respect to numerous directions (x,y), which is equivalent high in module proper values of matrix M. The arrangement of proper values is demonstrated in figure 2.

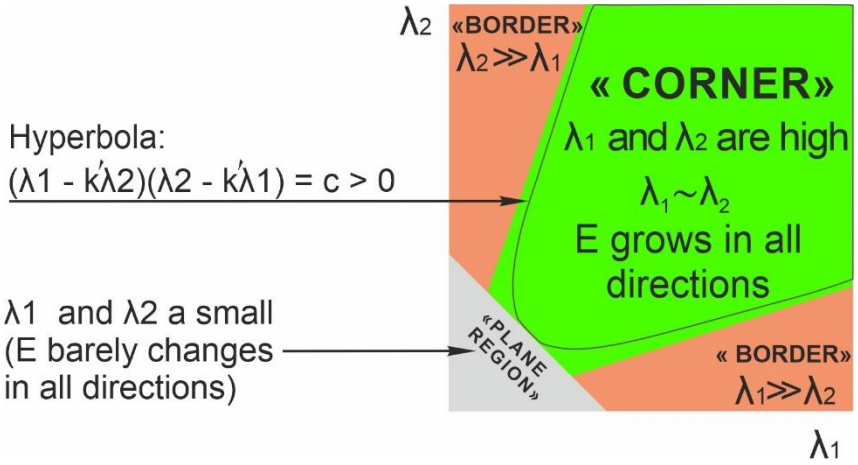


Figure 2. Changes in the values of function E (x,y) at corner detection.

Since direct calculation of proper values is demanding regarding the resources, the following response measure is suggested:

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

Thus, the values of such proper values determine, if the region is a corner, a border or a plane by the following criteria:

1. When | R | is small, which occurs, when λ1 and λ2 are small, the region is a plane.
2. When R < 0 at λ1 >> λ2 or vice versa, the region is a border.
3. When R is high at high λ1 and λ2 and λ1~λ2, the region is a corner.

If the value is above the predetermined threshold, then the point is deemed to be a corner and thus to be a point of interest. Figure 3 demonstrates that only when λ1 and λ2 are above the minimum value λmin, the point is deemed to be a corner (green region).

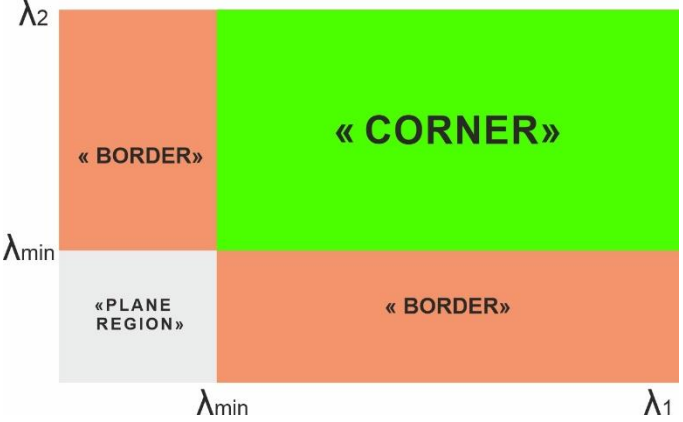


Figure 3. Determination of the values at corner detection.

3. Using OpenCV library

In order to make computer vision easier to use, OpenCV library was developed. This library contains functions that allows to process an image with a set of operations, without need to keep in mind of how images are stored in memory. Examples of these processes: get gray scale version of the image, resize an reposition part of images, filter images by colors, detect contours, and many other. Using this library enables a programmer to build a computer vision algorithm with ease. Because of library itself is written in C++, it is extremely efficient, but it doesn't require a programmer to use C++ language, as there are bindings to other languages. Bindings are libraries that delegate function calls from one programming language to another.

One of possibilities that this library provides is corner detection. It is introduced as two functions: “Harris corner detention” and “Shi-Tomasi feature tracker”. By the simplest possible definition, a corner is a point in space (two-dimensional in our case), where two edges are joined.

Harris corner detection algorithm is implemented as mathematical functions, that can be explained on high level. First, algorithm calculates a function which is able to determine in which direction the uniqueness of the window changes the most. After that, algorithm finds direction in which it should move the window to maximize uniqueness difference. It also find direction in which uniqueness changes the least. Having that, algorithm calculates, how much uniqueness change in both of these two directions.

At last, algorithm has to determine how different these values are, and how big they are, at the same time. When both values are small, this means that image almost doesn't change in any direction, which indicates a flat region. When one of values is much greater than the other, it means that by moving window in one direction changes uniqueness a lot, while moving in second direction doesn’t change uniqueness of the window at all, which, in turn, indicates an edge: moving along the edge doesn’t change uniqueness, moving across the edge changes uniqueness a lot. When both values are big, it indicates that moving in both of these directions effectively changes uniqueness of the image, which indicates a corner.

In order to calculate difference between these two values while also considering their size, algorithm must provide a scoring function, which returns a number denoting how much a certain part of an image looks like a corner. This is where Harris and Shi-Tomasi algorithms differ.

Shi-Tomasi algorithm suggests a different scoring function that is not that precise, but is a good approximation and is calculated much faster on computer processors. Speed is important because this function needs to be calculated for each pixel on the image. Function itself is pretty simple: it just takes the lowest of two values (figure 4):

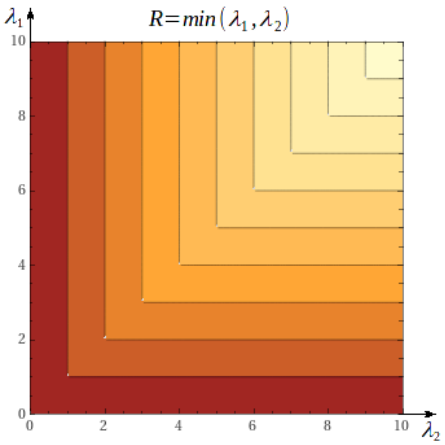


Figure 4. Plot for Shi-Tomasi algorithm corner scoring function.

Obviously, some information is lost due to such approximation. For example, values that correspond to flat regions are not that well defined now, and mixed along with values that correspond to edge regions. But because this algorithm is aimed on corner detection, such inaccuracy can be omitted.

In OpenCV library, Shi-Tomasi corner detection algorithm is implemented and wrapped into "GoodFeaturesToTrack" function. First function parameter is an object of a gray scale image. This function adds some logic to the algorithm. First of all, it picks N best corners detected on the image. Amount of desired comers to be detected is supplied as a second function parameter. Another words, it converts image matrix to a list of coordinates. Second, it applies score threshold value to remove worst corners from the list. Threshold value is supplied as a third function parameter. Third, this function is capable of removing corners that are too close to each other, and this value is supplied as a fourth function parameter.

Before image is processed by corner detection algorithm, it must be converted to gray scale color space. This is also done by one of the OpenCV function, which is capable of converting images between color spaces, considering gamma conversion and color intensities.

4. "GoodFeaturesToTrack" module operation

Modelling assistance is implemented in form of addon for Blender 3D. Belnder 3D provides a python library "Blender Python" ("bpy" for short). This library is imported into python code and enables a developer to interact with all Blender functions programmatically.

In order to create an addon, developer must implement a specific addon interface that is provied by bpy. This implementation should provide Blender with information about addon's name, description, type, author, and a functions to execute. In return, blender provides us a context object as a function parameter, which represesnts current state of everything in this Blender session. If these requirements are not satisfied, Blender refuses to import a script as an addon. After creating 3D corner points, the program tries where the walls of the building are.

```
class ImportImageWithCorners(bpy.types.Operator):
    """Script to track corners on a reference image""" # Tooltip
    bl_idname = "object.track_image_corners"
    bl_label = "Track corners"
    bl_options = {'REGISTER', 'UNDO'}
```

First step is to check if scene is set up properly for this addon to be used. Addon assumes that scene has an imported reference image, and this image is selected. Addon executes a couple of checks:

whether there's a selected object, whether this object is a reference object or not, does this object has a reference to image, and whether this image is imported correctly. Reference objects in Blender 3D are implemented as extended "Empty" objects. Empty object in blender is something that takes place in 3D space, but doesn't represent a 3D model. Image reference is one of possible types of empty objects. Even though that selected object might me an image, it might not have an actual image attached to it, so this must also be checked.

```
def execute(self, context):
    empty = context.active_object
    if not empty:
        self.report({"WARNING"}, "Nothing is selected!")
        return {"CANCELLED"}
    if empty.type != 'EMPTY':
        self.report({"WARNING"}, "This object is not an empty!")
        return {"CANCELLED"}
    if empty.empty_display_type != 'IMAGE':
        self.report({"WARNING"}, "This empty object is not an image!")
        return {"CANCELLED"}
    try:
        filepath = empty.data.filepath
        dimensions = empty.data.size
    except AttributeError:
        self.report({"WARNING"}, "Object has no img ref!")
        return {"CANCELLED"}
```

Second step, addon gets filepath to the image and loads it into the OpenCV library. It converts this image to greyscale, and executes corner detection function. This function recieves three parameters: number of corners, quality of corners and minimal distance between corners.

```
img = cv2.imread(filepath)
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
corners = cv2.goodFeaturesToTrack(gray,
self.corners, self.quality, self.distance)
if corners is None or not np.size(comers): return {'FINISHED'}
```

These parameters are also declared as addon's properties, so when a user executes the function, he's able to alter these parameters by entering values into the appeared dialog. Each value has either decimal or floating point type, maximum, minimum and step values:

```
corners: bpy.props.IntProperty(
    name="Number of corners",
    min=1, default=25
  )
  quality: bpy.props.FloatProperty(
    name="Quality of corners",
    min=0.00001, max=1, default=0.04
  )
  distance: bpy.props.IntProperty(
    name="Distance between corners (px)",
    min=1,  default=10
  )
```

When this addon is used, the following dialog appears to enable user to adjust values used for corner detection (figure 5):

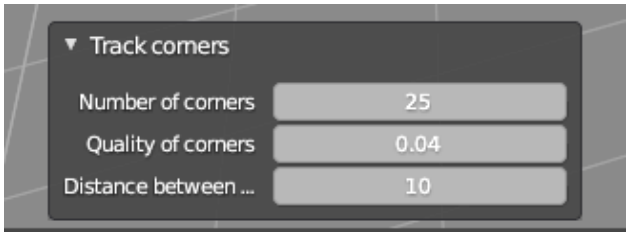


Figure 5. User interface for corner detection.

Corner detection function returns an array of coordinates, that represent corners in space of image's pixel coordinates, having point of origin at the top right corner. In Blender 3D space, for X and Y coordinates, in metric, and point of origin is historically at the bottom right corner. This means that corner coordinates must be mapped into the Blender space.

In order to do that, size of the reference image must be known. However, Blender 3D API only provides one dimensional size of the reference image. Through means of documentation and experiments, image was discovered to have its longest dimension align with the size of the dimension image. For example, when a square image is loaded, it's size is set to 5 by default, meaning at the image will be 5 units long and 5 units high, regardless of its resolution. If the image's width is two times larger than it's height, then in 3D space image will be 5 units wide and 2.5 units high. This means we will have to determine the bigger dimension of the image.

It should be also kept in mind that by default, image has offset to its origin point, 50% by height and 50% by width, making image's point of origin in it's center. This offset can also be read from Blender API.

The formula has the following form:

$$\boldsymbol{f(x) = \frac{(x/d_X + O_X) \cdot S \cdot d_X}{max(d_X, d_Y)}}$$

$$\boldsymbol{f(y) = \frac{(y/d_Y + O_Y) \cdot S \cdot d_Y}{max(d_X, d_Y)}}$$

In this formula, x and y are input coordinates, dX and dY are image dimensions in pixels, OX and OY is image offset in percent, S is size of the reference image. This formula maps into the following code, considering that additional dimensions provided by corner detection function should be stripped, and than each point should also have a zeroed Z coordinates as we map points from 2D space to 3D space:

```
e_size = empty.empty_display_size
offset = empty.empty_image_offset
dim = dimensions
f = lambda x: (x / dim + offset) * e_size * dim / np.max(dim)
corners = np.apply_along_axis(, 1, corners)
corners = np.append(corners * [1,-1], np.zeros((len(corners), 1)), axis=1)
```

Corners are now mapped into 3D space. However, these coordinates are local, meaning that transformations like moving, rotating or scaling, are not applied yet. However, this can be achieved by assigning a parent object to the created object. Blender 3D will automatically apply all transformations of the parent image to the created object.

During object creation, a couple of utility operations must be executed: mesh should be created and assigned to the new object, the new object mast be put into the main and default collection of objects, object must be selected by plugin convention:

```
mesh = bpy.data.meshes.new(mesh_name)
obj = bpy.data.objects.new(mesh.name, mesh)
col = empty.users_collection[0]
col.objects.link(obj)
obj.select_set(True)
bpy.context.view_layer.objects.active = obj
```

```
mesh.from_pydata(corners, [], [])
obj.parent = empty
return {'FINISHED'}
```

When object is created, parent object is assigned, and corners are imported into the mesh. Because we do not create edges or faces based on the vertices, empty arrays are supplied as arguments. At last, addon must return an exit code indicating that addon execution has gone successfully.

Due to current addon implementation, each time user changes one of the parameters, addon's main function is executed. It means that while user changes the slide input, this function is executed continually which can cause performance issues as the addon uses some performance heavy operations like reading file from disk into video memory. In order to improve addon performance, image should only be read once by introducing lazy static space for storing reference to this image, but this lies ot of scope of this project.

First, the user specifies the number of corners to be detected. Then the user specifies the quality level from 0 to 1, which means the minimum quality of the corner, all the corners bellow which are rejected. The minimum distance between the corners is ensured by Euclidean space. Upon specification of such information, the function detects corners on the image. All corners below the quality level are rejected, the function then sorts the remaining corners by quality from highest to lowest.

The function then takes the first strongest corner, discards all the corners in the vicinity within the range of the minimal distance and returns N strongest corners. Thereafter, the software places for each detected corner a primitive point, the vertex, in the three-dimensional space, which is within the specified building layout, and visually straightens with the layout corners. The user shall use it as a basement for grounding the individual building model.

Once three-dimensional corner points are created, the software tries to determine, where the building walls are. If the space on the line between two points is of dark color, the space is deemed to have the wall. The software creates borders between the points, where necessary (figure 6). Now, the user may interact with the created model: specify the thickness of all or of certain walls using modifiers, squeeze the building in vertical direction to reach the required height, place windows and door frames using logical operators or repeat processes for each building level. Thus, the exact model of any complex building may be created in less than a day, if the user has access to the layout diagram.

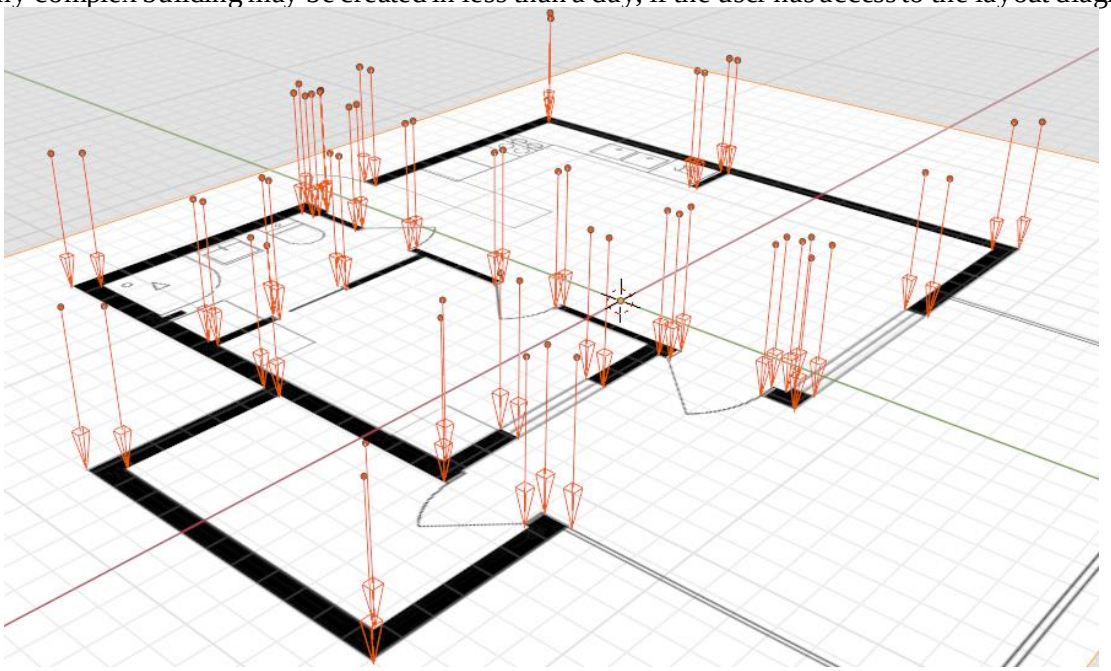


Figure 6. Results of operation of "GoodFeaturesToTrack" module in Blender 3D.

5. Conclusions

The conducted research allowed for design and development of the software module for performance of three-dimensional modelling in Blender 3D based on Shi-Tomasi corner detection algorithm. "GoodFeaturesToTrack" module is included in the developed BluePrint SDK and is one of the main modules of the mobile application of augmented reality.

Acknowledgments

The reported study was funded by State Research Programs Belarus, project number 20212185.

References

[1] Stefan Leutenegger, Margarita Chli and Roland Siegwart 2011 BRISK: Binary Robust Invariant Scalable Keypoints *ICCV* pp 2548-55

[2] Harris C and Stephens M A combined corner and edge detector 1988 In Fourth Alvey Vision Conference, Manchester, UK pp 147-151

[3] Aksionova N A and Kucharav A I 2021 Development of SDK for a mobile application using augmented reality technology *Bulletin of F. Skorina Gomel State University* **3(126)** pp 81-4.

[4] Varuyeu A V 2021 Information capacity of objects in conjunction with augmented reality *Bulletin of F. Skorina Gomel State University* **3(126)** pp 92-95

[5] Aksionova N A and Kucharav A I 2021 Presentation and implementation of educational material using augmented reality technology *Electronic collection of materials of the International scientific conference «Trends and prospects for the development of science and education in the context of globalization»* **71** pp 235-39