Augmented Reality: An Overview of Essential Algorithms

Sammy Lincroft, Emily Mattlin, and Allison Turner

CS 231-02

Bassem

20 Dec 2018

Intro

This paper provides an overview of approaches taken to the mechanics of augmented reality. Augmented reality is defined as applications that combine real and virtual elements in real-time, registered in three dimensions. The creation of augmented reality can be broken up into three tasks: recognition, tracking, and overlay (Schmalstieg). We will explore and discuss in turn algorithms for implementing each of these three tasks.

Background

Augmented reality has existed since Ivan Sutherland's first augmented reality system created in 1986 (Schmalstieg). The Milgram Continuum, conceived in the late 80's by Paul Milgram places augmented reality on a continuum between virtual reality in which a user experiences an entirely virtual world and the reality of real life. In the middle, in the state of mixed reality, lies augmented reality along with augmented virtuality (Trekk). Between 1998 and 2008 there were 313 papers published about tracking, interaction, calibration, mobile and other applications, evaluations, authoring, visualizations, rendering and multimodal augmented reality. In particular tracking, techniques were most studied with over 20% of published research exploring tracking algorithms (Zhou). Following this, interaction, calibration, and applications were the most studied fields with around 14% each (Zhou). This breakup of research illustrates the difficulty in recognizing and tracking the real world in order to create augmented reality and is where a large portion of algorithm research lies in the field.

Recognition

Augmented reality is achieved in three continuously cycling steps: recognition of the space and objects within the space, tracking motion of the objects, the space itself, and the

observer, and mixing in of augmentation elements (Amin 12). The first step, recognition, can be achieved in many ways. While visual data is undoubtedly valuable to augmented reality processing, as most augmentation involves the addition of visual elements or processing of visual elements, there are other forms of data that can be used to recognize spaces and objects. These forms include sonar and other forms of ultrasonic sensing, global positioning system data, accelerometer and gyroscope data, and many others, depending on need (Schmalstieg 112). There are also different types of visual data to consider, from straightforward pictures to structured light systems that register image depth (Schmalstieg 128).

The methods of recognition of space and objects can be divided into a few different categories based on what data they use, how they process it, and how they use the processed data. Fiducial marker-based tracking uses predetermined symbols to understand space. Using set symbols with very strong characteristics allows for fast decision making about the presence or absence of the symbol, especially when the markers are strongly colored as in the case of bitonal black-white fiducial markers (Hirzer 1). This method also is helpful for determining camera angle, since the size or warping of the marker can indicate distance, oblique angles, or similar (Amin 13). Hybrid tracking uses two or more data sources, such as GPS data, accelerometer data, or gyroscopic data, to recognize the environment that the user is in and where their camera is pointing. With the advent of services like StreetView on Google Maps, which is basically a virtual world model tied to GPS values, this data can be simply the final key in situating a user in a virtual world model and determining their perspective in order to decide what to augment in the user's field of view (Amin 13). A third way to approach recognition and situation of user environments is with recognition of real-world counterparts to 3D models. The programmer

creates 3D models of real-world objects before program use. The algorithm attempts to recognize lines and build polygons in order to compare its constructed 3D model of what it is perceiving with the predetermined model of the object or environment. This approach is very demanding of processing power, since it requires dynamic creation of 3D models and comparison of 3D models (Amin 13). The fourth category is natural feature tracking. This approach involves dynamic creation of models of certain pieces of the user's environment. These models are stored so that these objects can be recognized again later. This recognition model has the potential to be highly personalized to each user and to be very flexible with lighting, object orientation, occlusion, and other environmental interferences (Amin 13).

There are countless implementation details to consider in any method of object and environment recognition for augmented reality applications. So, consider as a guiding case study a technical report written by Martin Hirzer on a fiducial marker recognition system, with some detours into other studies for comparison. Hirzer's algorithm breaks down into five steps: choosing edge pixels, detecting edges, line extension, corner creation, and quadrangle building.

Deciding which pixels constitute an edge of an object is the first step in any visually based recognition algorithm. Volume of data can easily slow down this task, which can be solved by only analyzing full images every so often and analyzing only a sampling grid of the pixels on most frames as well as exploiting image locality by assuming objects are not moving very far between frames (Hirzer 5-6). In Hirzer's case, selection of edge pixels comes down to a threshold color differentiation decision on the pixel itself in the context of its neighbor pixels, since he is looking for edges on black-white fiducial markers. This decision is easily complicated in more complex markers, natural feature tracking and model-based tracking.

A different edge pixel determination system, can be found in Piotr Dollar and C. Lawrence Zitnick's structured forest approach to image analysis. Dollar and Zitnick state that "[p]atches of edges exhibit well-known forms of local structure, such as straight lines or T-junctions" (Dollar 1) and use these known structures to create decision trees. These trees branch off based on what structure the patch may best fit, and yield a designation of edge pixel or non-edge pixel for the pixel(s) in the center of the patch. This decision tree based algorithm is better suited to a natural feature tracking based approach to recognition because of its flexible definition of an edge or edge pixel. However, Dollar and Zitnick's algorithm is not completely described by the decision tree model, since "[i]ndividual decision trees exhibit high variance and tend to overfit" (Dollar 3). Dollar and Zitnick thus tune accuracy by labeling pixels with the cumulative decisions of a virtual forest of decision trees, all diverse enough from each other on the images that they use as their samples that the result is very accurate (Dollar 3).

A more common method of edge pixel selection is the Canny algorithm, which can be summed up in the following pseudocode:

"1. Smooth the image with an appropriate Gaussian filter to reduce desired image details.

2. Determine gradient magnitude and gradient direction at each pixel.

3. If the gradient magnitude at a pixel is larger than those at its two neighbors in the gradient direction, mark the pixel as an edge. Otherwise, mark the pixel as the background.

4. Remove the weak edges by hysteresis thresholding." (Ding 1)

Gaussian filters are commonly used in graphics to blur images and apply other effects, using a pixel and its neighbors to contribute to varying degrees to an output pixel's final shade. A more

efficient implementation of a Gaussian blur runs at O(n) time, n being the number of pixels in an image, since it samples colors horizontally, n constant time samplings of k pixels averaged in a constant time Gaussian function to contribute to each of the n pixels, and then vertically in a similar fashion, for a total of 2n applications of the Gaussian function (Rákos 1). These steps are constant time comparisons or decisions for each of n pixels, making Canny edge detection an O(n) operation on an image with n pixels. After the removal of weak edges, the Canny edge detector only returns strongly detected edge pixels or pixels above a certain threshold and connected to edge pixels, indicating them as more weakly showing edge pixels.

Once edge pixels have been decided, Hirzer defines four categories of line detection algorithms to choose from: hypothesize-and-test statistically based algorithms, algorithms based on gradient magnitude and direction, local contouring of groups of edge pixels, and Hough transform based approaches (Hirzer 4). Hirzer opts for a statistically based algorithm because he does not want to sacrifice the considerable processing time for a Hough transform based algorithm, despite its accuracy. The statistically based algorithm that Hirzer chooses basically draws a trend line using the edge pixels and their common orientations, their orientation being defined by their situation along a gradient.

The Hough transform is a particularly interesting algorithm based in linear algebra. It is made to fill an accumulator array whose dimensions are x θ-values and y ρ-values within a certain range and spaced to a standard difference. The transform fills each cell with “votes” for the θ-values and ρ-values indicated by its placement in the array. Votes are counted by which edge pixels fall along the imaginary line parameterized by the specified θ-values and ρ-values. Θ-values are the angles between the normal to the imaginary line, the normal which passes

through the origin, and the x-axis, and ρ-values are the lengths of the sections of the normals between the imaginary line and the graph origin. The lines that have the most votes will be decided to be real, and the edge pixels that support the existence of that line will be taken out of consideration for other lines (Yam-Uicab 1). Below is some pseudocode for implementation of the Hough transform:

"1. Obtain I_b, result of binarizing *I*.

2. Quantize parameter space (ρ,θ) into accumulator cells $M[\rho,\theta],\rho \in [\rho_{min},\rho_{max}]$; $\theta \in [\theta_{min},\theta_{max}]$.

3. Initialize all cells to 0.

4. For each foreground point (x_k,y_k) in the thresholded edge image I_b:

- For each point θj equal all possible θ-values
  - Solve for ρ using $\rho=x_k\cos\theta_j+y_k\sin\theta_j\rho=x_k\cos\theta_j+y_k\sin\theta_j$
  - Round ρ to the closest cell value, ρ_q
  - Increment $M(p, q)$ if θ_p results in ρ_q

5. Find line candidates where $M(i, j)$ is above a suitable threshold value.

6. Return lines $\rho_i=x\cos\theta_j+y\sin\theta_j\rho_i=x\cos\theta_j+y\sin\theta_j$" (Yam-Uicab 1).

The initialization of the algorithm, i.e. steps 1 to 3, takes some multiple of km steps, where k is the number of ρ values being considered and m is the number of θ values being considered. Step 4 is bounded by the number of edge pixels n, and then by the number of θ values being considered m, because of the consideration of all θ values for each edge pixel, making the voting part of the algorithm O(mn). The arithmetic steps within are constant time simple operations.

Steps 5 and 6 require another iteration through the accumulator matrix, taking another km steps. In total the algorithm seems to run at O(km + mn) time.

Once lines have been identified, through whatever method chosen, they can be extended to their full lengths to intersect with others and create corners and full polygons. If they can't be extended on either end, it is assumed that this was a false positive of a line and it is deleted, since lines must always intersect with other lines in some fashion in the real world. Hirzer describes a process in which the color differentiation surrounding a certain pixel can be tested to see if the line should be extended in that direction. A certain allowance is made for extending the line with pixels diagonal to the previous end of the line in order to allow for lines with inconsistencies and slight curves (Hirzer 11). Once lines have been extended far enough, they will intersect with others and create corners. It is necessary to check the orientations of the lines in order to ensure that all intersections make sense; no parallel lines should intersect, and a line shouldn't intersect with itself (Hirzer 13). Once valid corners are constructed, they can be connected with lines in common, making a quadrangle - a polygon. All models, however complex, are made of polygons, and so this process can be scaled up to detect hundreds, thousands, millions of polygons, and to identify real-world counterparts to virtual models.

Tracking

Once polygon coordinates have been established for the real world, the next step is tracking. Tracking refers to a dynamic registration of coordinates in order to track the relative movements of real-world objects and the augmented reality system (Schmalstieg). There are three main categories of tracking. Sensor-based tracking utilizes magnetic, acoustic, inertial, optical and/or mechanical sensors to determine position. The more recently prevalent

vision-based tracking utilizes image processing technology to determine positions. Hybrid-based tracking uses a combination of sensors and images to track objects positions. For example, combining inertial tracking data with vision-based tracking can help to accommodate fast movement where vision-based tracking often fails. Over 80% of recent research has been on vision-based tracking which is where we will focus our attention. (Zhou)

Within vision-based tracking, natural feature tracking utilizes image processing techniques to find lines and edges as described in Martin Hirzer's algorithm. In Ulrich Neummann and Suyu You's paper Natural Feature Tracking for Augmented Reality, they describe a system for closed-loop motion tracking and demonstrates the advantages of using a natural feature based tracking system. While Neuummann and You's paper also delves into methods of identifying natural features, we will focus on their tracking methods which could be applied to key points or regions selected by any number of recognition algorithms.

At the most basic level, tracking the motion between frames requires determining the motion between each image. This is done by repeatedly computing the inter-frame motion v where $v = A^{-1}B$ and A and B represent the coordinates of key natural features at two close points in time. This approach assumes a linear or translation motion of each natural feature. However, in the case of sets of features or the shape of a region as a whole, there may not be simply a translational component and an affine transformation that takes into account rotation and scaling may be necessary. This affine warp can be calculated in the same way as the inter-frame motion. To calculate the effectiveness of a transformation or the error in the detected transformation, we compute region C where C = Av, or the transformed first region. We can then compare this transformed R_c to the actual second region B or R_t. This gives us the motion residual

$\varepsilon = \frac{\left|\left|R_t(x,t), -R_c(x,t)\right|\right|^2}{MAX\{\left|\left|R_t(x,t)^2, R_c(x,t)^2\right|\right|\}}$ and tracking confidence $\delta = \frac{1}{1+\varepsilon}$. If the tracking confidence drops below a certain threshold, image recognition can be re-applied in order to identify better features to track and thus improve tracking confidence. This closed-loop system and method of built-in error checking can prevent re-computing natural features on every frame improving efficiency.

Unfortunately calculating A^{-1} still requires inverting a matrix, an operation which is $\Theta(n^3)$. While still polynomial time, this calculation must be performed between every frame in order to ensure smooth motion tracking in realtime which is a primary reason why visual tracking is so computationally demanding.

Once reliable transformations between frames have been established, it is possible to determine which transformations must be established. There are three main transformations used in augmented reality. Visual tracking is responsible for establishing two categories: model transformation and view transformation. Model transformation is responsible for establishing the relationship between moving objects and global world coordinates. For example, a car driving across a crosswalk on video. View transformation establishes the crucial relationship between eye coordinates or camera coordinates and global world coordinates. By keeping track of where the camera is in space, it is possible to keep consistent the location of the global world coordinates which allows applications to add virtual objects into the real world and keep their motion independent of the motion of the camera. This allows for the final transformation perspective transformation which combines these virtual and real-world objects into a single output display.

Overlay

Once the real world has been tracked and registered, we must overlay the virtual scene onto the real scene. When the virtual scene is projected onto the real scene, there are many different types of displays and elements to be generated depending on the type of interaction you want for the user. Each one of those elements has to appear natural and blend into the real scene as seamlessly as possible; the visual registration has to be clean. This means that depth of field and placement has to be taken into account as well as lighting and shadows. The first step in displaying the virtual images is to do a projective transformation of the 3D camera coordinates that have been tracked and recognized and map them to the 2D device display. The view frustum processed by the camera is mapped onto a unit cube and then the z coordinates are dropped. The x and y coordinates then are transformed using the screen's units in the correct aspect ratio and displayed. Of course, not everything in augmented and mixed reality is supposed to be blended seamlessly into the user's environment! What fun would that be? A perk of augmented reality is that the displays don't have to be mapped to just the world around you; there are displays that can be fixed to the camera's (the user's) perspective.

One of these types of objects that is fixed to the camera is a screen-stabilized element that is always in the user's display. This is useful if you want a menu or a clock or maybe a logo constantly in view. Screen-stabilized elements are fixed to the camera, and so are body-stabilized elements, but those are not always in view. Body-stabilized elements are fixed to the camera, but really just move with the user as he moves. So if you have a 3D avatar in the virtual world, that image is a body-stabilized element; the body of the avatar isn't constantly in view, but when you walk around, it follows. Virtual objects mapped onto a user's real world are called world-stabilized elements. For example, if a box was generated to sit on the floor in front of you,

it wouldn't matter if you turned around, walked past the box, looked up or down because the box will always stay in the same spot on the floor. Body-stabilized and world-stabilized elements don't have to be rendered unless they are within the view frustum, which costs less for the GPU. In order to make the box look realistic and natural, the real and virtual scenes must be registered.

This visual coherence is achieved by generating depth cues from the camera and organizing virtual objects onto the 2D screen to appear as if it blends with the 3D world. An important depth cue is the relative size of an object because when something is farther, it looks smaller. Similarly, farther objects appear to have be higher up in the view. Shading cues give insight into where light sources are depending on how an object is lit and so virtual objects can be lit properly and cast shadows. Also, there are occlusions, which is when a closer object obscures or covers whatever is behind it. Virtual objects that are generated to be closer to the user occlude objects behind them, but if a virtual objects needs to be more distant from the user, any object closer than that virtual object will occlude the virtual object. The distances of these real objects are generated from depth cues.

A cohesive environment consisting of the real and virtual will only be convincing if occlusions are properly resolved: objects must appear to be where they are meant to be placed. Occlusion culling is "the process of removing objects that are hidden by other objects from the viewpoint" (Mayer 1). In most practices, occluded objects aren't rendered, which lessens the GPU's rendering load. We will touch on three different types of occlusion techniques and algorithms: Potentially Visible Sets and Occlusion Queries.

The Potentially Visible Sets register areas seen from the camera that are not within the current view frustum, but have potential to be viewed. This process is time consuming and is

done before the virtual scene is displayed. The perk is that rendering is much better after the Potentially Visible Sets are completed. This technique is mostly very useful if there are no live changes to the environment during the user's interaction. Because this is performed once, before the user interaction, if there are dynamic changes to the real world, the display will not adapt.

Occlusion Queries are gaining popularity and it is common for it to be incorporated into graphics hardware. Because this technique is built into the hardware (using depth sensors and tracking in the camera), it's very simple to use in a rendering algorithm because the technique is already implemented. For this method a query is made for a real object and its boundary and the GPU responds with the amount of pixels needed for the virtual object to be rendered in the display. If the object's pixel amount is very low, then the object doesn't have to be rendered because it is too occluded by a real object. This method allows for using the real scene more in the display, allowing for more interactivity now that any real object can occlude a virtual one. This process may cause lag because queries are continuously being made each frame to check on the boundaries of the real objects. However, the real scene can change and the display and rendering will react.

There are many more algorithms and methods that have been used to handle occlusion culling, but fine-tuned versions of occlusion queries seem to be the current and future winner. With that bright future of more advanced occlusion culling, there also are tools to use it for! Displaying and developing augmented reality programs have never been this popular and accessible.

Tools

Many developers are using Unity to power their augmented reality games as they have for virtual reality. Since AR is such a hot topic right now, many new SDK are being published and released. Some of these include Vuforia, ARToolKit, Google ARCore, Apple ARKit, Maxst, Wikitude, DeepAR, EasyAR, Xzimg, and argon.js. Besides smartphones, hardware for augmented reality is quite inaccessible due to its high price tag; the current leading AR headset, the Magic Leap is currently being sold for about $2,300 and the Microsoft HoloLens is about $3,000.

Applications

Augmented reality is applicable in countless areas. What first comes to mind are uses like Pokemon Go and self driving cars. Pokemon Go is easily pinned as a good use of hybrid tracking, and the augmentation is easy to see: a cute little Pikachu hopping around your local baseball field, or an Eevee in your front yard. Self driving cars combine hybrid tracking with natural feature tracking in order to navigate the real world, avoid hitting objects, buildings, people, other cars, identify parking spaces, and control speed. However, augmented reality has significant importance outside of the world of video games and convenience.

Augmented reality has a lot of potential in the realm of making everyday life more accessible for disabled people. “A system called ASRAR is created to help a deaf person, which shows what the narrator says as a text display to deaf people. System combines augmented reality with, automatic speech recognition and TTS technology to help people to communicate with deaf persons without use of sign language” (Amin 14). This project allows deaf people to operate with ease in any context. It means they are not at the mercy of others’ abilities to use sign language or use pen and paper. Another project seeks to improve the experiences of those in

wheelchairs while shopping. A model of a store was created where each shelf was indicated with a fiducial marker, and RFID scanners were placed on the back of each shelf. A disabled shopper would usually not be able to interact with most of the items in the store, but with this system, they can hold up their phone or tablet camera to the fiducial marker, the RFID scanner will retrieve data on which items are currently on the shelf, and the user will be able to browse metadata - read summaries and reviews, browse as an able-bodied shopper would - on the available items without having to struggle to reach them (Rashid 1).

Another project improves the quality of oral surgery by recognizing features of a mandible and superimposing medical scan data onto teeth, gums, and jaw. Oral surgeons have years of practice and training, but visualizing nerve paths, tooth roots, and areas of infection helps them make more precise choices when performing a surgery. This would be a perfect application for model based tracking, since each tooth is unique and has ample CT scan imaging available. The researchers developed an algorithm to translate the tooth into the 3D model and back with informational overlay (Murugesan 1).

Conclusion

Computational understanding of our world has incredible potential and infinite possibilities. The surface has only been scratched. With the rising capabilities of CPUs and GPUs and progressions of versions of these algorithms, augmented reality will make our world computationally understandable and accessible for everyone. So now we know how to use it and have ideas of what to it for, but will we incorporate headsets into our daily life? Will virtual entertainment replace real world interactions? According to critics, it already has.

Bibliography

Amin, Dhiraj, and Sharvari Govilkar. “Comparative Study of Augmented Reality SDKs.” International Journal on Computational Science & Applications, vol. 5, no. 1, 2015, pp. 11–26., doi:10.5121/ijcsa.2015.5102.

Clemens, Arth, Raphael Grasset, Lukas Gruber, etc. “The History of Mobile Augmented Reality Development: Developments in Mobile AR Over the Past 50 Years.” Inst. for Computer Graphics and Vision. Graz University of Technology, Austria. May 6, 2015.

Anderson, MJ. “Augmented or Virtual: How Do You Like Your Reality?” *Trekk*, Trekk, 22 June 2015, www.trekk.com/insights/augmented-or-virtual-how-do-you-your-reality.

Ding, Lijun, and Ardeshir Goshtasby. “On the Canny Edge Detector.” Pattern Recognition, vol. 34, no. 3, Mar. 2001, pp. 721–725.

Hirzer, Martin. “Marker Detection for Augmented Reality Applications”. Graz University of Technology, Oct 2008.

Kalkofen, D., Sandor, C., White, S., and Schmalstieg, D. “(2011) Visualization techniques for augmented reality”. In: Furht, B., ed., Handbook of Augmented Reality, Springer.

Mayer, A. Julian. “Dynamic Occlusion Culling.” TU Wien, TU Wien, 2007, pdfs.semanticscholar.org/5cab/0fac256b81b71c8d85a51881f7daf52ab7f3.pdf.

Murugesan, Yahini Prabha, et al. “A Novel Rotational Matrix and Translation Vector Algorithm: Geometric Accuracy for Augmented Reality in Oral and Maxillofacial Surgeries.” *The International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 14, no. 3, 2018, doi:10.1002/rcs.1889.

Nair, Bindu, et al. “Fast Edge Detection Using Structured Forests.” *International Journal of Emerging Trends in Science and Technology*, 2016, doi:10.18535/ijetst/v3i08.06.

Rákos, Daniel. “Efficient Gaussian Blur with Linear Sampling.” *RasterGrid Blog*, RasterGrid Blogosphere, 7 Sept. 2010, rastergrid.com/blog/2010/09/efficient-gaussian-blur-with-linear-sampling/.

Rashid, Zulqarnain, et al. “Using Augmented Reality and Internet of Things to Improve Accessibility of People with Motor Disabilities in the Context of Smart Cities.” *Future Generation Computer Systems*, vol. 76, 2017, pp. 248–261., doi:10.1016/j.future.2016.11.030.

Schmalstieg, Dieter, and Höllerer Tobias. Augmented Reality: Principles and Practice. 1st ed., Addison-Wesley, 2016.

Technologies, Unity. “Occlusion Culling.” Unity - Manual: State Machine Basics, 2018, docs.unity3d.com/Manual/OcclusionCulling.html.

Yam-Uicab, R., et al. “A Fast Hough Transform Algorithm for Straight Lines Detection in an Image Using GPU Parallel Computing with CUDA-C.” The Journal of Supercomputing, vol. 73, no. 11, 20 Nov. 2017, pp. 4823–4842., doi:10.1007/s11227-017-2051-5.