Crowd-sourced Data Collection Pipeline for Human Attention Tracking in Virtual Environment

Zhiyang He
Stanford University
hzyjerry@stanford.edu

Abstract

Attention is a very useful metric in understanding how human beings explore novel scenes. In recent years, machine learning community has adopted this mechanism to make significant progress in computer vision and natural language processing. With the rise in VR platforms and devices, it becomes an intriguing question how human beings explore virtual environments in comparison to 2D images and texts. Good understanding of human attention can cast insight on robotics and 3D computer vision. In this project, I extended effort of former researchers[5] to record human attention in virtual reality, and proposed a mobile-based VR attention tracking pipeline that can crowd-source data collection process. Source code of the project is published[1].

1. Introduction

In recent years' researches in computer vision and robotics, attention-based mechanism has become a highly active field. Compared to traditional vision methods that take entire images as input, attention based learning offers several advantages:

First, they it has better adaptation to human visual system: human eyes do not accept every pixel simultaneously as equally accurate. Our foveated vision chooses a sub-area to focus, while our brain chooses the next area to look at. Convolutional Neural Networks oversimplify this feedback-control mechanism as input-output system.

Second, attention mechanism offers lower computational cost: CNN runs slow on large images because the computation on 2D image is linearly correlated with image resolution. Attention-based learning allows us to aggregate local information and improve the speed of visual algorithms.

In this section we illustrate these advantages with actual usage of attention mechanism.

[1] Project is maintained at https://www.github.com/hzyjerry

1.1. Attention in Natural Language Processing

Before used in computer vision, attention is first applied in natural language processing to solve translation problem.

Traditionally, machine translation based on deep learning are implemented using recurrent neural network. We map the meaning of an input sentence in foreign language into a fixed length input vector, and then decode an English sentence based on the vector. Neural network offers many advantages over hand-engineered features like n-gram, and leads to significant performance boost. But it seems unreasonable to assume that we can encode the meaning of a potentially very long sentence into a simple vector.

With attention mechanism we no longer try to encode the full source sentence into a vector. Instead we allow the decoder to "attend" to different parts of the sentence at each step of output generation. A big advantage of attention is that it gives us the ability to interpret and visualize what the model is "looking at":

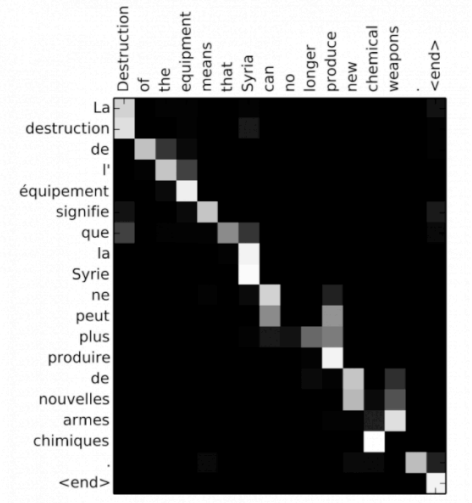

Figure 1: while translating from French to English, the network attends sequentially to each input state

1.2. Attention in Computer Vision

In computer vision, similar method from NLP is borrowed to offer better insight into what the system is doing. In Show, Attend and Tell, attention mechanism is applied to image description to show which area of the image is focused on by the neural network to generate every word.

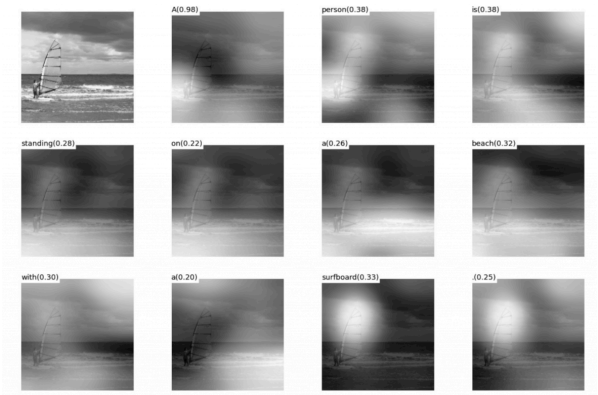

Figure 2: Image from Show, Attend and Tell, the output of the agent is "A person is standing on a beach with a surfboard"

1.3. Attention in Virtual Reality

Deep learning tasks are less well-studied in virtual environment. But with the rise of virtual game platforms and indoor scanning data made available, using machine learning technique beyond 2D data has become a very active field. Stereo reasoning has a lot of applications in robotics because tasks like scene classification, 3D object detection are extremely useful for navigation and autonomous driving.

Here I argue that attention mechanism will prove highly useful for machine learning beyond 2D data, because (1) the amount of stereo image has much larger volume than 2D counterpart, and the information richness will be more unevenly distributed in input, (2) most tasks inside virtual environment do not require global information about scenes, but rather visual cues of local areas. For example, to know where to go next, a robot shouldn't need image from behind it.

For these reasons, generating saliency map in virtual environment is highly valuable. In the next section I will discuss existing works on visual saliency prediction and VR human attention tracking.

Here we need to note that human visual saliency though similar, is still different from the attention demonstrated by AI agents that we mentioned in part 1. For example, we do not look at a different spot in the image for every word we speak. Despite this, human visual attention is still useful for different reasons. If we acquire enough data on human visual saliency, we can train machine learning algorithms to predict saliency on novel images. Even if the data is not enough for training, human visual saliency can be very good oracle baseline for us to compare machine generated visual saliency.

2. Related Work

Saliency prediction on 2D images is a fairly well studied problem. There exist several well-known benchmark and datasets for this problem. Saliency map on virtual images, in comparison, is a lot less explored. Some researchers have argued that visual saliency in virtual environment demonstrates distinct properties that 2D saliency no longer applies.

2.1. 2D Saliency Map

The most well known saliency benchmark is proposed by MIT lab[2]. Over the past years, machine learning algorithms have achieved close to human baseline accuracies (92%).

The most widely used data set for visual attention is the SALICON[3] data set. The data is collected in a crowd-sourced manner, using mouse as a proxy for eye movement. Despite it's size, the collection is solely based on participants moving their mouse, which is lacking in convincing that the saliency map truly represents what human eyes focus on. Most of the other datasets are based on similar approach where participants are asked to click on the things they see.

In my project I propose a recording mechanism that is more accurate and reliable than mouse tracking.

2.2. VR Attention

Stanford researchers (Sitzmann et al.) collected attention information for human in VR environment using eye-trackers[5]. They collected 780 head and gaze trajectories using DK2 head-mounted display and pupil-lab stereoscopic eye tracker. Due to the complexity of set up process, the dataset collected is not enough for conducting saliency prediction or other machine learning tasks.

The goal of my project is to propose a solution that can bridge the gap between the need for visual data and the difficulty in collecting. I will build an end-to-end data collection system, where I use human head orientation in virtual scene as an approximation for eye direction in order to collect human visual attention under specific VR tasks like scene understanding. To scale up data collection, I rely on low-end Google Cardboard and Amazon AMT to deploy the application.

[2] http://saliency.mit.edu/
[3] http://salicon.net/

3. Design and Implementation Details

3.1. How to Approximate Eye Trajectory?

Tracking the exact focus and movement of human eye is a very difficult task. Existing solutions are either expensive or require significant amount of work to set up, which imposes significant difficulty in conducting large scale data collection for human visual attention.

In this project we use head orientation to approximate human eye attention. This is based on the assumption that in virtual environment, human rely more on head movements to explore areas outside their foveated region, instead of eyeball movements. Former study[5] have confirmed the validity of this approach.

By using head orientation as a proxy for eye orientation, we largely reduce the cost of setting up our pipeline. This is the major assumption and innovation of this project.

3.2. Task Specific Attention Tracking

In former work[5], researchers have found that human random explorations inside virtual environment do not have strong tendency to converge. This can be explained by the fact that users with different purpose focuses on different things. For example, someone looking for a table inside a virtual scene will notice different visual cues (floor, chairs) from another person searching for the lights.

In order to collect more purposeful human movement, we will present participants a certain task at the beginning. For example, a user will be asked to specifically look for all the tables inside a room. By formulating the task, we can further categorize the data we collect, and compare the convergence of attention map across similar tasks.

3.3. Head Movement

In modern mobile web browsers, device movement can be detected using eventHandler[4]

```
window.addEventListener("deviceorientation", handleOrientation, true);
```

Despite differences in handling orientation across browsers[4], cross-platform javascript plug-ins for reading device angle are available and well-tested.

In our application, we record three different values simultaneously: (alpha, beta, gamma), and upload them in real time through Socket.IO. The definition of these angles is specified in Web API:

```
alpha: rotation rate along the axis perpendicular to the screen

beta: rotation rate along the axis going from left to right of the plane of the screen

gamma: rotation rate along the axis going from bottom to top of the plane of the screen
```

3.4. Visualization

There are many different ways to visualize human focus. In this application I use Raycaster to render focus points as red dots on the virtual scenes. Different from expectation, the visualization requires non-trivial modification of existing libraries, mostly due to ThreeJS incompatibility issues. This is also the majority of the work in this project.

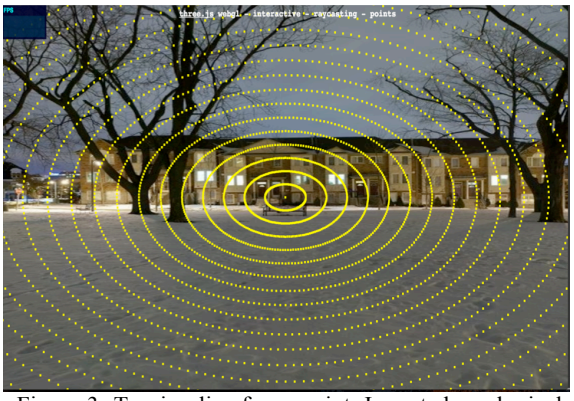

Figure 3: To visualize focus point, I created a spherical point cloud and superimpose it on top of the background. The intersection of eye direction and the sphere is enlarged. The points on the sphere are later changed to transparent using vertex shader.

The Raycaster is modified from the above example. First I remodel the point cloud model to create a sphere, and modified its position to center it around the camera. I changed the size of the points and the animation to make it more visually observable. Then I superimpose the sphere and stereo scene background so that the enlarged focus point will be directly displayed on top of the background. After this step, I implemented vertex shader inside render script to make the sphere transparent, so that the background will not be blocked.

The visualization result is illustrated in the figure below:

[4] https://developer.mozilla.org/en-US/docs/Web/API/Detecting_device_orientation

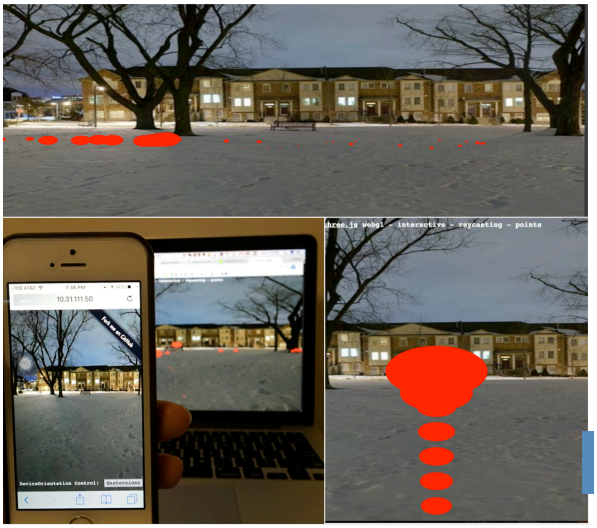

Figure 4: Top: eye trace of user scanning the environment in horizontal direction. Bottom left: working scenario, where the user is able to enter vr environment on mobile. Bottom right: the positions gazed by the user for a long time will be emphasized with big red dots.

4. Project Component

In this section, I describe the set up process, as well as architecture of the application. In addition, I discuss the potential cost and viability of deploying this application on AMT platform.

4.1. Requirement

The main project server is implemented in Javascript front end with Nodejs backend. The data streaming part of the project is written using Socket.IO. In order to run it and develop on local computer, one needs to have the latest Nodejs (v6.5.0) installed.

4.2. Local Set up

The app will be eventually deployed on AMT platform, and the app server will be running in the cloud so that people across the world can access it. But in reality, one will likely need to test out the app on local computer, and modify or develop it. This section provides instruction for setting up the app and accessing it using mobile phone. The entire set up should take less than 5 minutes.

The entry point of the project is index.js. One can start up the server by going into the project folder and executing 'node index.js'. The Nodejs app will start running on localhost at port 5000. After this step, two entry points will be created: (1) localhost:5000/vr is used for participants to explore the environment, and (2) localhost:5000/visualize is used for researchers to examine visual attention in real time.

Note that entry point (1) fetches IMU data provided by web browser on mobile device, thus if the app is run it inside computer browser, an exception will be raised saying no compass is detected. In order to use mobile device to access the same entry point, one need to make sure that the computer and mobile device are in the same network in this scenario. After this is ensured, one can simply access the web app by typing in compterIP:5000/vr on mobile web browser. Note that the computerIP can be found by running 'ipconfig', or by executing the following command inside node command line tool (with ip node module installed):

```
var ip = require('ip');
console.dir(ip.address());
```

4.3. AMT Set up

To set up and deploy the app on Amazon Mechanical Turk, one needs to follow the AMT manual to set up HIT jobs, task specifications and task prices. In order for Mechanical Turkers to understand the task, one also needs to provide an UI with task specification using HITLayout, and a way for Turkers to access the web app on their mobile phone. To increase flexibility, this UI is not included inside the source code. But here I propose one way to instruct Turkers to use computer and mobile device:

(1) Step 1: Mechanical Turkers click into the Task page, reads the task instruction (what to do inside VR scene, how much time is given, etc), and gets informed to prepare a mobile device and a google cardboard.
(2) Step 2: After Mechanical Turker clicks start task, the Turker will be given a short link to open on mobile browser, and a 6-digit access code which can be typed into the web page. Once the code is typed in, a VR scene related to that code will be displayed, and the task and timer will be started immediately.
(3) Step 3: Once the Mechanical Turker finishes the task, the webpage on the Turker's mobile browser will display another 6-digit code, which can be typed back to AMT interface on computer, to indicate that the task is finished. Beside asking for the return code, the AMT task website will also ask a couple questions related to the VR scene (e.g. write a sentence to describe what you saw in the scene).
(4) After step (1) - (3), the task is finished and the Turker will receive payment.

4.4. Web Application Architecture

In this section I discuss the architecture of the web application. The purpose is to show the scalability and low cost of setting of the application.

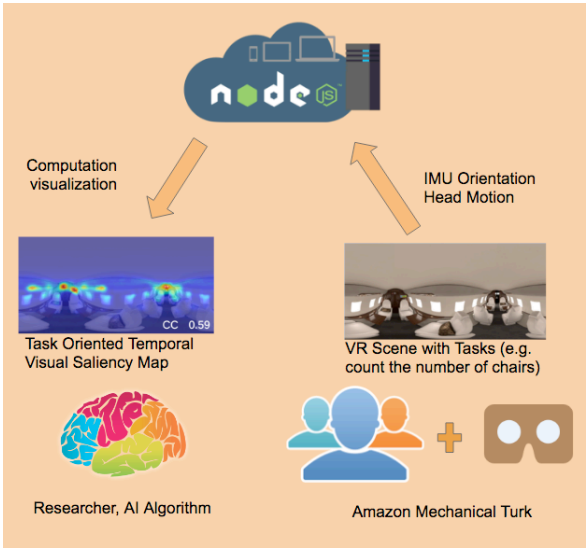

Figure 5: workflow of web-based data collection and visualization

Our VR render, IMU movement stream, database and visualization are all backed on NodeJS server deployed in the cloud. This means that users all across the world have access to the AMT task, and researchers can visualize the data globally, without physical constraint.

As the user look around in the VR scene, his/her head orientation is recorded by web browser, and sent to server in real time. The server stores the data stream inside its database. To visualize, researchers can either stream the data in real time, by polling readings on Socket.IO, or by fetching a previous record from database to visualize in a video-like manner.

We use browser-enabled VR display, and do not require users to install any third party app (like Google Cardboard App) or plug in. The app can be instantaneously opened on mobile browser, and once the task is done, users can close the page and not leave any trace in their mobile phone. Although browser-enabled VR leads to couple problems that will be discussed in part 4, this design largely reduces the time needed for tasks, and makes it friendly for first-time Turkers on this task.

4.5. VR Tasks

In our task set up, we want the Turkers to perform purposeful tasks, as opposed to randomly exploring the scene. Thus before the Turker sees the VR scene, we will provide a task prompt. Here are some exaples:

(1) Find out all the household utensils in the room. You do not need to memorize them.
(2) Find out what time it is during the day, by observing the environment.
(3) Observe the scene, and try to make some guesses about the owner of the place.
(4) Observe the scene, and try to guess where it is, and what is it used for.

After the Turker completes the task, a prompt will be given to type in answers with regard to the task. This answer can also be used to double check that the Turker put in effort to inspect the scene.

4.6. Cost

There are two main sources of cost for the data collection: (1) payment for AMT workers, (2) remote hosting fee. For AMT payment we can use standard task fee, and increase the payment accordingly depends on how much we want to speed up the collection. For web app hosting, we can rely on existing services like AWS hosting.

Since we do not need to provide eye tracking hardware, and participants do not need to be physically present in the lab, the cost of our pipeline is much cheaper than former solutions. This comparison is shown in the chart below.

	Lab-based eye tracking (Sitzmann et al.)	AMT-based eye tracking (ours)
Device Cost	Pupil-lab Eye Tracker: 1800 Oculus DK2 Head Mount: 400	Server Deployment: <30
Cost Per Test	Lab Participants: 15/hr	AMT Workers: <0.1/task
Set Up	Device set up and calibration every time	No set up needed
Parallelizability	One experiment per time	Highly parallelizable

Chart 1: comparison of lab-based eye tracking and AMT-based eye tracking

In addition to the low cost, this application can be run 24/7 on AMT platform and do not require researchers to be present with participants. This greatly reduces the amount of work for researchers.

4.7. Viability

Different from former researches, this project does not rely on sophisticated eye tracking tools, and has almost zero set up time. The only dependency is that the participants need to have a mobile device and a google cardboard.

By latest news in March[3], Google has already sold over 10 million Google Cardboard globally. Although this is still far from being widely-available, the owners of Google Cardboard (teenagers, collage students, active web users)

have high overlap with participants of Amazon Mechanical Turk. This means that deployment of the task on AMT is likely to lead to prospective yield.

In addition to using AMT, the data collection task can also be done in a lab setting, where Google Cardboard is provided, and participants are invited and paid on hourly basis. Even in this case, our pipeline has several major advantages. First we do not need to set up VR head mount and eye tracker, which largely reduce the workload of researchers. Second, instead of being limited by the number of expensive eye trackers, we can easily scale up the experiment by deploying dozens of eye tracker in the lab, and have many users participate at the same time. This can help speed up the data collection speed, and shorten research cycle.

5. Conclusions

This project presents an economically feasible approach to scale up data collection of human VR attention.
However, there are several development issues that I didn't expect while designing the project. In this section I discuss these difficulties, and the implications they have for future development. I also list the future work at the end of the section.

5.1. Threejs Incompatibility Issue

One of the biggest issue that came across to me while implementing this project is the incompatility issue in Threejs. To enable a project that uses both 3D rendering, stereo effect, IMU and attention visualization, both early and late version of Threejs have blindspots. It is hard for one to avoid dealing with Threejs source code and making a customized version. In my final implementation, I used three different versions of Threejs, to support the functionality. The major incompatibility issues are:

(1) ThreeJS before release 60 do not render properly on mobile device. Current browsers on mobile devices do not provide backward-compatibility for old ThreeJS versions. However, most available animation plugins for ThreeJS are implemented 2~3 years ago using outdated versions. These libraries did not take into consideration of mobile rendering when they were originally implemented.

(2) ThreeJS after release 80 are banned on mobile web browser for IMU access. This is because mobile browsers like Safari and Chrome want to protect user privacy and prevent random website from accessing user movements. This has strong implication in Threejs VR community. After the ban, most developers abandoned VR development in browser, and moved to mobile application instead.

(3) ThreeJS stereo effect does not support directly loading stereo images. Our implementation uses square images instead of stereo images to render virtual scenes. Since stereo image is the standard format for visual saliency map, the application cannot directly compute visual saliency.

(4) ThreeJS changes its API for stereo effect and ParticleSystem after release 70. The attention visualization of this project is implemented using libraries implemented with ParticleSystem API. However, ThreeJS made several drastic changes in their API that causes the library to break (ParticleSystem was renamed to Points, and later to PointCloud). Backward compatibility is not implemented in latest versions.

These incompatibility issues cause the existing Threejs libraries to be extremely fragmented. Most plugins are implemented using different versions, and few of them work with each other without causing Threejs breakdown. This makes it really hard to incorporate different rendering features together.

5.2. Potential Problems

As is mentioned in the former section, the incompatibility issue of ThreeJS imposes difficulty in implementation. Although our application is enough to support simple visual exploration inside virtual environment, it is questionable whether it can be extended to support more interaction tracking (moving, integration with HTC Vive, etc.) in the future.

Another concern is user privacy. In latest release of ThreeJS, readings of user IMU data inside mobile web browser is prohibited. This problem has not arisen in earlier versions of ThreeJS and we're still able to obtain IMU data using outdated ThreeJS library, but it is unsure whether browsers like Chrome and Safari will complete block any access to IMU on mobile devices. One solution to this is to obtain HTTPS domain for our application, and certificate to access user data, but this will lead to much more overhead work.

5.3. Future Work

There are several directions that can help further refine the project.

(1) Defining meaningful virtual tasks and find valid data sets to perform these tasks. The tasks listed in section 4.5 is only a small subset of all possible tasks.

(2) Real-time Saliency Map computation. In this project, the human attention is abstracted as a red dot in the virtual scene. Saliency map computation is not included in the implementation. Future work

can add in plug-ins to calculate Saliency Map on virtual image using IMU stream.

(3) Extend Stereo Renderer in ThreeJS to support stereo images. As is mentioned in 5.1(3), our current implementation uses square images instead of stereo images.

(4) Improve mobile compatibility. The application is only tested on safari on IOS devices. Since it's unpredictable what device Mechanical Turkers will use to perform the task, the application need to perform well on all mobile platforms using different browsers.

References

[1] Attention and Memory in Deep Learning and NLP, wildml, 2016

[2] D. Bahdanau, K. Cho, Y. Bengio, Neural Machine Translation by Jointly Learning to Align and Translate, ICLR, 2015

[3] Google Has Shipped Millions of Cardboard Virtual Reality Devices, Fortune, 2017.03

[4] Detecting device orientation Specification, MDN Mozilla Developer Network

[5] V. Sitzmann, A. Serrano, A. Pavel, M. Agrawala, D. Gutierrez, G. Wetzstein, Saliency in VR: How do people explore virtual environments?, Arxiv, 2016.12

[6] Authors. The frobnicatable foo filter, 2006. ECCV06 submission ID 324. Supplied as additional material eccv06.pdf.

[7] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. Zemel, Y. Bengio, Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, Arxiv, 2015.10