International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 08 | Aug - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

3D Modelling for the Hajj and Umrah Pilgrims

Dr (Er) Om Prakash

Professor SMS, Lucknow

ABSTRACT

A 3D simulation model is generated that can be used for a virtual representation of people who are performing Hajj and Umrah so that future travelers can gather the necessary information to perform religious ceremonies and be well prepared before arrival to the holy places. Traffic and pedestrian management in the areas is also presented. The purpose is to speed up the calculation of all events of interest to the authorities in a simulator, such as the behavior of vehicles, pedestrians on the streets, and worshippers. Two GPU solution alternatives and one CPU solution alternative are proposed, and the performance of the proposed solutions is also compared.

Introduction

In recent years the creation and use of different types of simulators have increased; either as a means of prediction and understanding of natural phenomena (Tsunami, earthquake or hurricane simulators) or as training tools for system operators (driving, flight, train simulators), where the objective is clear; model the behavior of a complex system in the real world. In similar lines, a 3D model can also be prepared to simulate the arrival, stay, and departure of pilgrims for the Hajj and Umrah. To achieve a higher level of accuracy and complexity, some simulators use a micro-simulation model that allows them to operate at the level of the individual. However, this requires a high level of computational processing, which can only be achieved with parallel processing (Mohamed et al., 2019).

The simulation model proposed in this document seeks to create a tool that allows showing possible scenarios to which the transport system may be exposed. To achieve this, geographic data from the work areas of the transport system have been taken and a micro-simulation model has been implemented that takes advantage of the GPUs by performing parallel processing, which allows micro-simulations to be carried out at speeds greater than current CPUs. This paper shows some alternative solutions that have been developed in some simulators. In section 3 the simulator rules are defined and several solution alternatives are proposed, in section 4 the performance of the various proposed solutions is compared, and in section 5 future work is shown (Mohamed et al., 2019).

Approach

A 3D model based on an IoT-based framework is used which is not only smart but also efficient in terms of crowd time management. Mohamed et al. had proposed a similar system and we can use and elaborate such a system. This approach permits users to interact with mobile devices and there is an interface layer that utilizes the various sensors and the data from these devices. The data flows to the management layer. This converts the data to information that provides us with vital information about open roads and passages. This then provides us with somewhat non-crowded areas as it binds data to user groups and friends. The expert system manages crowd monitoring using this IoT-based framework. A prototype was also proposed by Nasser et al. which is designed to predict possible problems by monitoring the paths leading to the location of the rituals.

The proposed 3D model will also enhance the proposed model by Islam et al., who presented an IoT-based Crowd congestion and stampede avoidance approach that uses a combination of different sensors and machine learning-based WEMOS D1. The prototype was known as E-writs belts was designed to collect data from pilgrims in real time and predict the possible risk of a stampede. The proposed 3D model using the sensors and connected devices will use the past data, current movement of the pilgrims, the data interchange amongst the users, and the mandate to stay connected to the devices will not only count the current number of pilgrims just before the Haj but will use the past data and predictive analytics to precisely predict the actual numbers. The user interactions and profiles will help to identify the people around Haram and count their number. IoT-based model has been successfully used in similar data collections as proposed by Nasser et al.

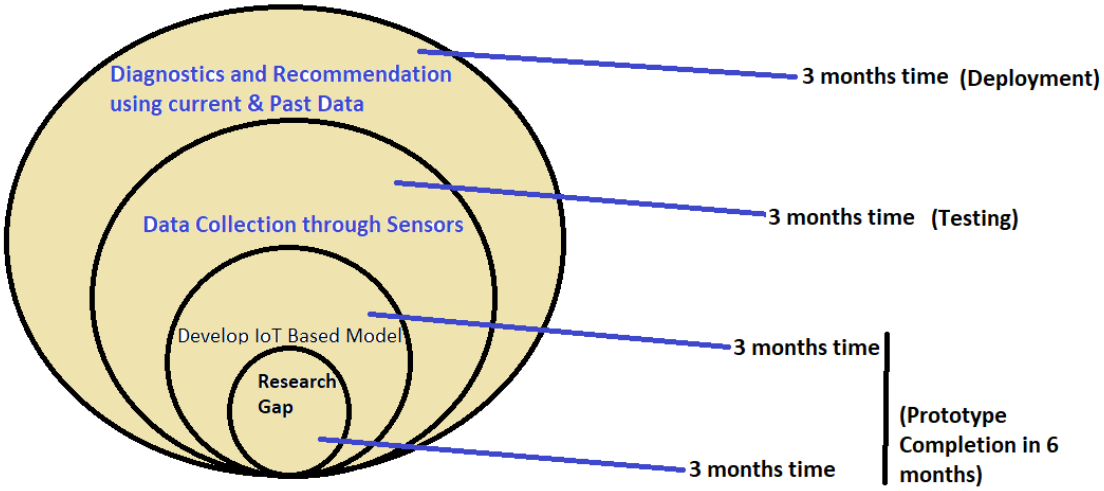
The same model as proposed above will estimate the flux, density, and count at a given instant. The same model will also predict the density of pilgrims at a given time at a given place. Both predictive and prescriptive analytics will be used to reveal the exact statistics and recommend the best course of action to avoid congestion and provide suitable diagnostics to the organizers. The current and acceleration of people in various directions will be calculated (Mohamed et al., 2019).

Roadmap:

The following roadmap is identified to detail the schedule of events and milestones forecasting and communicating the planned solution deliverables over the time horizon. A visual tool is also used to assist for developing and communicating planned deliverables & milestones concerning time to highlight the type of work currently being undertaken at the point in time. The object of developing this roadmap is to offer the team the ability to develop, evolve, and adjust the activity being planned. Besides, this roadmap will also provide the stakeholders with a bird's eye of the current activity being undertaken and the overall long-term deliverables. Following is the breakup of the activities envisaged in the complete development plan for the proposed and offered solution:

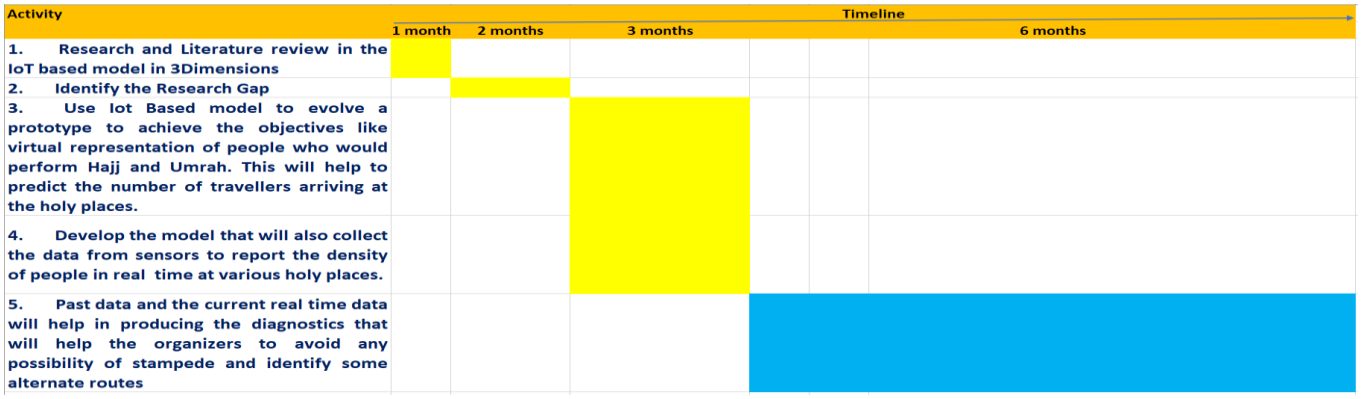
1. Research and Literature review in the IoT-based model in 3Dimensions
2. Identify the Research Gap
3. Use Iot Based model to evolve a prototype to achieve the objectives like virtual representation of people who would perform Hajj and Umrah. This will help to predict the number of travelers arriving at the holy places.
4. Develop a model that will also collect the data from sensors to report the density of people in real-time at various holy places.
5. Past data and the current real-time data will help in producing the diagnostics that will help the organizers to avoid any possibility of stampede and identify some alternate routes
6. A total of 6 months will be needed to complete the prototype. A 3-month will be needed for testing the model and another 3 months to deploy the model and commissioning.

The above tasks are proposed to be completed in a year. The Planning Horizon is represented below (total 1 year time needed for deployment):



Planning Horizon: Total 1 year time needed for deployment of the solution

The flowgraph of the development and deployment is also represented belos:



Flowgraph depicting the development and deployment of the proposed 3D model

Research and Literature Review

Simulation models are computer models that operate at the level of individual behavior, such as people, families, or firms. Such models simulate large representative populations of these entities to draw conclusions that can be applied to higher levels of aggregation; an example may be the average speed of a vehicle in a city. This type of model is different from aggregate models whose explanatory variables already represent the collective properties. In the corpus of current literature, some models of simulators were presented (Islam et al., 2019).

Address fields can be drawn by users or extracted from a video sequence; the system represents these fields in the free spaces of the simulation environment. Because agent trajectories are implicitly encoded in these fields, they must meet the following conditions (Islam et al., 2019).:

- Agents must create paths of the least effort to reach their goals.
- Fields must be able to pass around static objects in the environment.

Researchers have also, for the representation of the direction and navigation fields, used the free space of the environment. It is discretized, through a grid, where each cell has a stored direction vector, a free or busy state, and handles a connection of 4 neighbors.

Another similar solution is presented where there is a micro-simulation model based on a discrete simulation space. Here space is subdivided into small regions, and each region has defined mobility parameters. This space allows to represent structural pathologies such as the state of the road network. Movable entities (pedestrians and vehicles) can occupy one or more regions of the space, the definition of their displacement is defined by the average of the mobility parameters of the regions they occupy (Islam et al., 2019).

3D Model Based on CUDA architecture.

CUDA (Compute Unified Device Architecture) is the parallel computing architecture developed by Nvidia that tries to exploit the advantages of GPUs over general-purpose CPUs by using the parallelism offered by its multiple cores and allowing the launch of a high number of simultaneous threads (Jason and Edward, 2010). CUDA programming, serial operations are still handled by the CPU (main processor), while parallelizable 'Kernels' are handed over to the GPU for processing. It is important to understand the CUDA architecture and memory design. Each Kernel is assigned a Grid; each Grid contains a series of Blocks; and each Block contains a set of threads. It is possible to have a maximum of 8 active blocks per stream multiprocessor or a maximum of 24 active warps (a warp is a set of threads that are physically executed in parallel) per stream multiprocessor. With 32 threads per warp, we have a maximum of 768 active threads per multiprocessor stream. On a card like the GTX285 we would have 23,040 active threads (Sachine, et.al., 2010).

Implementation

The Implementation plan needs to complete the five activities from undertaking the Literature Review to completing and deploying the 3D Model The details of the implementation are as follows:

The Literature Review and related activities were already described in the previous section. In this section, firstly the next activity, which is IoT IoT-based 3D model is presented, along with the necessary details.

The 3D simulation model is based on the Cellular Automata model. A cellular automaton is a discrete model containing a collection of cells within an n-dimensional grid of known size, the cells "evolve" through several discrete time steps according to a set of state-based rules. neighboring cells and the state of the cell itself. The model has 3 types of entities: Pilgrims (or pedestrians), Vehicle,s and Holy Places (Sachine, et.al., 2010).

The rules that entities must follow are:

- Entities try to reach a global goal (a destination), through one or more local goals (like Holy Places).
- No entity can occupy a cell that is being occupied by another entity.
- “Pedestrian” type entities can pass through any cell except those that represent buildings or red traffic lights for pedestrians.
- “Vehicle” type entities can only pass through vehicular roads, and also can only move in the direction that the road allows (see section 3.3). Likewise, these entities must stop at red traffic lights for vehicles.
- “Holy Places” type entities must be at their Holy destinations according to a defined time. The local goals of these entities represent an arrival/departure route and the approach stations that their route covers (Sergio, 2010).

Texture Map

The developed model requires that the geographic space be discretized in a grid of fixed size. This is done through a texture file, where a pixel represents a cell, and the color of each pixel represents the state of the cell itself in the 3-Dimensional space. For the creation of the texture file, the geographic information of the simulation area is taken (the Holy places) and encoded in a texture map. The codification of the texture map has been done taking advantage of the RGB space with which the color of each pixel of the image is represented.

- Layer R (red): the information on the type of cell (building, pedestrian street, vehicular street) and entities that occupy the cell (Holy Places, vehicle, or Pilgrims) are encoded.
- Layer G (green): the mobility parameters of the zone are encoded (directions in which the entity can move)
- Layer B (blue): traffic signals (traffic lights) are encoded.

One pixel in the texture represents an area of approximately 1-meter square. In addition to the texture map, a series of local and global goals are defined, which due to their complexity are not encoded in the texture map. Each local goal has its identifier and identifiers of the closest local goals in the north, south, east, and west directions, therefore an entity that is in a local goal can take 1 of the 4 available routes to reach the next local goal. A global goal is the initial (source) or final (sink) destination of an entity. An entity must pass through different local goals to reach a global goal. The path an entity takes to reach its global goal is calculated before runtime (Sachine, et.al., 2010).

Consistency Management: In CUDA

The active threads are executed in parallel, this leads to advantages in terms of processing time but also leads to disadvantages in terms of consistency, since one of the rules defined in section 3 (rule 2) specifies that 2 features cannot occupy the same cell. An example of movement interaction between 2 pedestrians, it can be seen that 2 pedestrians try to occupy a cell that is not being occupied by another entity at an instant of time, if both pedestrians proceed to occupy the cell in question, there will be a conflict because 2 pedestrians are occupying the same cell. Because of this, it is necessary to develop a conflict detection and correction stage, which is executed every time an entity moves (each simulation step). In the below section, two alternatives are shown to detect and correct these conflicts (Jason and Edward, 2010).

Development of the 3D Model

In this current sub-section, the development of the model that will also help in the collection of the data along with the usage of sensors is presented:

In this part, the development of the simulation step has been proposed, two in GPU and one in CPU, which are explained below (Sergio, 2010):

Simulation Step with Serial Conflict Detection and Resolution (GPU)

This solution executes the simulation step of each type of entity (Pedestrian, Vehicle, and Holy Places) in 6 phases (Sergio, 2010):

i. speed control phase,
ii. passage phase to the next cell,
iii. serial conflict detection phase,

iv. problem resolution phase,
v. conflicts in parallel, a second phase of detection of conflicts in a serial way and
vi. the phase of the resolution of conflicts in serial. With this solution, it is expected that a large

Part of the conflicts will be resolved in the first 4 stages and the final stage, which resolves the conflicts serially, will be executed for a very low number of entities.

Phase 1. Speed control (Parallel): To vary the movement speed of the entities, the concept of segments has been created. An entity belongs to only a single segment but a segment can have many entities. Only one segment can be executed in a simulation step, the more speed a segment has, the more often its simulation step is executed, in this way the speed of movement of the entities is controlled. For the creation of segments, a numerical identifier is assigned to each entity, and by numerical order a segment is assigned.

The number of segments created depends on the velocity assigned to the entities, the lower the velocity of the entities the more segments are generated, the higher the velocity the fewer segments are generated. If you work at maximum speed, a single segment is created that is executed in all the simulation steps.

Phase 2. Passage to the next cell (Parallel): In phase 2, the movement to adjacent free cells is carried out in parallel for those segments that must carry out their simulation step; however, as shown in section 3.2, this can lead to conflicts. In this phase, the following simplified code is executed, which is executed in parallel for all entities in the active segment (Jason and Edward, 2010):

```
for (each entity in the active segment){ findNextCellBreak();
        occupyNextCell();
}
```

Phase 3. Conflict detection (Serial): Conflicting entities are detected as a result of phase 2.

```
for (each entity in the active segment){
        for (all other entities in the active segment){
                if(there are other entities occupying the same space that I occupy)
                        checkInConflict();
        }
}
```

Phase 4. Conflict resolution (Parallel): In this phase, conflicts between the entities are resolved in parallel. This phase makes the entities evaluate a new position for their displacement; Of course, when carrying out this phase in parallel, the entities can enter into new conflicts (different from the initial conflict) (David and Wen-mei, 2010).

```
for (each entity in the active segment and marked with conflict){
        if(among the conflicting entities, this one has the lowest id)
```

```
            deletePreviousPosition(); // this implies that this entity takes precedence and takes ownership of the conflicting cell
        else // for all other entities
            findNextCellBreak();
    occupyNextCell();
}
```

Phase 5. Conflict detection (Serial): Conflicting entities are detected, as a result of phase 3 (the implementation is the same as that carried out in phase 2).

Phase 6. Conflict resolution (Serial): Conflicts between the entities detected in phase 5 are resolved serially, this phase causes the entities to re-evaluate a new position to carry out their displacement, however, as this phase is carried out in a serial, conflicts between entities are not presented again (the implementation is the same as the one made in phase 4, with the difference that it is processed serially). This solution effectively corrects the conflicts caused by the execution of steps in parallel, however, it has many stages executed serially, so the virtues of CUDA are not used correctly. Due to this, a second alternative solution was proposed by GPU.

Simulation step with parallel conflict detection and resolution (GPU)

This solution executes the simulation step of each type of entity (Pedestrian, Vehicle and Transmilenio) in 4 phases: speed control phase, passage phase to the next cell with conflict detection, conflict resolution phase in parallel with detection of new conflicts, and final conflict resolution phase. Phase 1 has the same algorithm as the one explained in section 3.4.1, for this reason, its explanation will be omitted.

Phase 2. Passage to the next cell with conflict detection (Parallel): In this phase, the same procedure is carried out as in phase 2 of section 3.4.1, with the only difference that, in addition to the texture map, it is used a map of entities, where the identification number of the entity is written in the cell that you want to occupy (David and Wen-mei, 2010).: }

```
For (each entity in the active segment)
{
    findNextCellBreak();
    occupyNextCell();
    writeToMapOfIDs();
}
```

In writeToMapOfIDs(), the entity identifier is written to a map similar to the texture map. The idea of using an additional map is to allow the detection of conflicts in parallel, in figure 9 you can see the procedure, in (a) can see how the behavior with the texture map would be; two entities with IDs 8 and 20 try to occupy the same free cell, and conflict, in (b) these two entities try to write their identifier in the same cell of the entity map, however as the

execution is in parallel, one entity will necessarily overwrite the data of the other entity, leaving (c) a single identifier in the cell.

Now to detect conflicts, each entity must review the identifier that has remained in the cells of the entity map. If the identifier written in the cell does not correspond to your identifier, it means that you have had a conflict with another entity; Otherwise, one of two things could happen: it did not conflict with another entity, or it did conflict but has priority when taking the cell, and the other entities are the ones that will have to recalculate their positions (German, 2010).

Phase 3. Conflict resolution (Parallel): In this phase, conflicts between entities are resolved in parallel, similar to phase 4, with the difference that the entity map is used to detect conflicts and writing in the map of entities the new positions are taken.

for (each entity in the active segment){

if(cell in entity map has the same id as the entity)

deletePreviousPosition(); // this implies that this entity has no conflicts or has precedence and can take ownership of the cell

else // for all other entities

findNextCellBreak(); occupyNextCell(); ***writeToMapOfIDs();***

|

Phase 4. Final conflict resolution (Parallel): In this phase, conflicts between entities are resolved in parallel, since this phase is carried out in parallel, any new movement can generate conflicts, for this reason, no new movements are generated, entities that do not have priority (whose id is not in the entity map cell) are returned to their previous position which is consistent and without conflicts since their position had not been released yet and the entities with priority finish taking ownership of the new cell by deleting its previous position (German, 2010).

for (each entity in the active segment){

if(cell in entity map has the same id as the entity)

deletePreviousPosition(); // this implies that this entity has no conflicts or has precedence and can take ownership of the cell

else // for all other entities

returnPreviousPosition();

}

It is important to note that in this solution alternative there are no serially processed phases, so the CUDA architecture is better used.

3D Simulation step without conflict detection and resolution (CPU)

Because the CPU processing is carried out completely serially, in this version there are no conflicts between entities, so the simulation step is carried out in only 2 phases: the speed control phase and the phase of passing to the next cell. The implementation of these phases is the same as that shown in the previous section with the difference that their processing is done serially

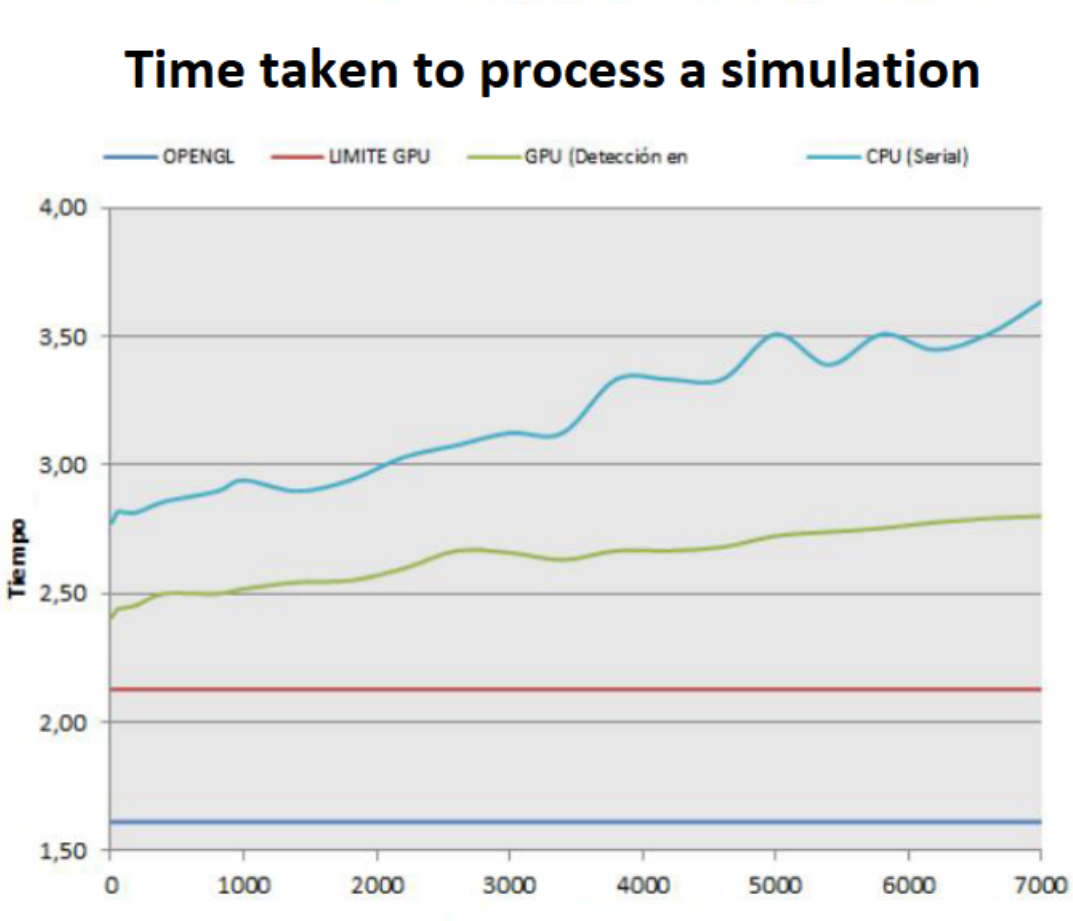
RESULTS The solution

Alternatives presented showed different results in the time taken to perform a simulation step, these results are shown below.

The time required to carry out a simulation step for the pedestrian-type entity.

It can be seen that the solution with parallel conflict detection requires less time to carry out a simulation step when the number of entities is high. and that the time remains relatively constant (initial of 2.41ms and final of 2.8ms). Performing linear regression, it is obtained that the time increment per entity is approximately 0.0560useg/entity and 0.6542useg/entity for the parallel and serial conflict detection solutions, respectively. This means that parallel clash detection increases simulation timeless as the number of entities increases (German, 2010).

To compare the performance of GPU processing against CPU processing, the times of the solution with parallel conflict detection for GPU have been taken and compared with the processing times on CPU. ***The results are presented in the Graphical form below:***



Graphical Presentation of Results

Discussion on Results

The graph of the GPU (green line) and CPU (blue line) processing time, along with some lower limits that cannot be exceeded. To understand these limits, it must be taken into account that the texture map is displayed using OpenGL (purple line) and that if no changes are made to the texture, OpenGL takes around 1.61ms to redraw the texture on the screen, therefore this is a lower limit that cannot be exceeded by either the GPU version or the CPU version.

Likewise, for the other limit, a kernel has been made to update the texture map by GPU (red line), where each pixel of the texture map is updated by a GPU thread and this has a set of minimum instructions, this update takes around 2.13ms so the GPU version cannot go faster than this value (German, 2010).

The simulation step time for the GPU version is lower than the CPU version and as the number of pedestrians increases the simulation step time increases more for the CPU version than for the version on GPU. When performing linear regression, it is found that the time increment is approximately 0.0560useg/entity and 0.1228useg/entity for the GPU and CPU version respectively; this makes the GPU version more suitable for a larger number of entities. Therefore, the GPU solution with parallel conflict detection and correction is the most efficient of the solutions shown.

CONCLUSIONS

- The implementation of the simulator based on CUDA has shown good processing times, especially when detecting conflicts in a parallel and non-serial manner.
- The GPU version is significantly faster than the CPU version, even though the CPU version is done with only one phase for the simulation step and not with 4 phases like the GPU version.
- Although the simulator does not have a large number of traffic parameters, it is a good approximation that can be expanded to satisfy other types of functional requirements.

References

- M. F. Mohamed, A. E.-R. Shabayek, and M. El-Gayyar, IoT-Based Framework for Crowd Management. Cham, Switzerland: Springer, 2019, pp. 4761.
- N. Nasser, M. Anan, M. F. C. Awad, H. Bin-Abbas, and L. Karim, ``An expert crowd monitoring and management framework for Hajj," in Proc. Int. Conf. Wireless Netw. Mobile Commun. (WINCOM), Nov. 2017, pp. 18.
- S. Islam, A. Ka, M. Z. Islam, N. Islam, and M. N. Ullah, ``IoT based crowd congestion and stampede avoidance in Hajj using wemos d1 with machine learning approach," in Proc. 4th Int. Conf. Electr. Inf. Commun. Technol. (EICT), Dec. 2019.
- Sachine Patil, Jur van den Berg, Sean Curtis, Ming Lin, Dinesh Manocha, (2010). Directing Crowd Simulations Using Navigation Fields. IEEE Transactions on Visualization and Computer Graphics.
- Sergio Arturo Ordonez (2010). microphone platform Scalable and Multimodal Simulation to Evaluate Urban Mobility in Unconventional Scenarios. Colombia
- Jason Sanders, Edward Kandrot (2010). CUDA by Example - An Introduction to General-Purpose GPU Programming, Ed. Addison-Wesley
- David B. Kirk, Wen-mei W.Hwu (2010). Programming Massively Parallel Processors, A Hands-on Approach, Ed. Morgan Kaufman.
- German A. Florez (2010). display development realistic for urban environment on Nvidia Scenix. University of the Andes Library, Colombia