3D Battlefield Modeling and Simulation of War Games

Baki Koyuncu, Erkan Bostanci

Abstract—The aim of this study is to develop a software tool which will help commanders to view the battlefield in real time for decision making processes. The battlefield was modeled by a terrain model which was generated from real world data. Military units were displayed on this terrain by using both recent military symbology and 3D models of the military units. Position updates for units were animated by using translation animation techniques. A database was designed to store data about military units existing in the software for later review and long term strategies.

Keywords—3D Battlefield Visualization, Sandbox, Military Decision Making, XNA, Database, MS SQL Server.

I. INTRODUCTION

In today’s modern warfare, the uncertainty of battlefield is much greater than it was before. Battlefields have spread to very wide areas of ground and the importance of military intelligence has increased tremendously. Consequently classical map techniques, in which a printed map of the battlefield was used to view the battle-space, had proven to be insufficient.

Commanders needed [1] efficient visualization systems to view the most recent data about the current military situations in order to make efficient decisions that would lead to victory. With these needs in mind, a virtual military sand-box software was developed for modeling and simulation of battle field and war games.

II. DEVELOPMENT

Development of the virtual military sandbox was divided into two main parts. First part was the graphical system which was used to view the objects corresponding to real world military units. The second part was the database which was used to store a military situation with all its object data.

A. Graphical System

A Window’s forms application was developed to visualize the battlefield. In order to draw the battlefield and the military units XNA was employed [2]. XNA is a graphics framework and is used with C# programming language.

1) Basic View

In this study, military units were displayed in the battlefield by using military symbology named NATO-APP-6-A [3]. This symbology included the necessary information for each military unit such as unit type, affiliation, position information and report date and time as given in Fig. 1. In a previous implementation simple geometric shapes were used for modeling military units and it was concentrated more on the efficient visualization of the terrain [4].

Battlefields were modeled with the data obtained from USGS web site in DEM format. A specific file-parser class named DEMFile was implemented for this type of data. This parser was inherited from a more general Elevation-Data class which included the general information about regional elevation data and elevation limits.

2) Inclusion of military models

Additional 3D models of the military objects were added next to the symbology used in the basic visualization introduced in [5]. In this way, the symbology became clearer and easier to understand and the visual quality of the developed sandbox software was improved.

Before the models were added to the software, they were edited in 3D Studio Max and their polygon numbers were reduced using MultiRes modifier [6] with minimum sacrifice from the actual model detail. These models were later displayed as symbols in the battlefield in the basic software.
converted in FBX format which is one of the formats supported by XNA.

Size differences arising in the modeling process were reduced to acceptable levels before they are displayed by the software. An important size difference was identified as the scale inconsistency. For example, a jeep model can be few times greater than a tank model if no scaling is applied to one of them or both. This difference exists since the modelers do not use a common scale while generating their 3D models. Appropriate scale levels were introduced by bringing several models together and identifying their relative sizes.

In addition, a default heading vector for all models was considered as the positive Z axis (0, 0, 1). This heading vector was used in the rotation of the models according to their new positions entered by the user.

These processes had to be performed manually for both single and groups of models which constituted one of the most time consuming parts in the study.

A class named Shape3D was designed to manipulate the 3D models. This class worked as an element of the Entity class. It stored the necessary transformation information in order to view the model in the exact position entered by the user. Major classes used in Sandbox are summarized as shown in Fig. 2.

![Fig. 2 Positions, heading vectors and rotation angle](image)

**Fig. 2** Positions, heading vectors and rotation angle

Given the initial heading vector and the new heading vector, the dot product between the two vectors is given by:

\[ a \cdot b = |a||b|\cos \alpha \]

where \( |a| \) and \( |b| \) denote the lengths of the vectors \( a \) and \( b \). \( \alpha \) is the angle of rotation. Then \( \alpha \) becomes:

\[ \alpha = \arccos \left( \frac{a \cdot b}{|a||b|} \right) \]

The following code segment shows how this formula is implemented in the software:

```csharp
position2D = new Vector2(position.X, position.Z);
newPosition2D = new Vector2(newPosition.X, newPosition.Z);
newDirection2D = Vector2.Subtract(newPosition2D, position2D);
cosAlpha = Vector2.Dot(newDirection2D, heading2D) / (newDirection2D.Length() * heading2D.Length());
alpha = MathHelper.ToDegrees((float)Math.Acos(cosAlpha));
```

Once the rotation angle is obtained, the software needs to know in which direction the rotation will be applied. This is simply decided by checking the X coordinate of the new position. If this value is positive, the rotation will be in clockwise direction and the rotation will be in opposite direction otherwise.

After these calculations, the model will be aimed to the new position as shown in Fig. 4.

![Fig. 3 Aiming the model to the new position](image)

**Fig. 3** Aiming the model to the new position

### 3) Determination of an object’s new directional vector

An object has an initial position and a vectoral motional direction as shown in Fig. 3. When a new positional data for this object is given by the user, the object has to rotate to head to this new position. This rotation will result a realistic animation of the models when the models are translated to their new positions.

The angle required for rotation is calculated by using dot product between two vectors. These motional vectors are shown in Fig. 3.
4) Position update animation

In [5], selected units appeared directly in the new position when an update is performed. This can be acceptable for viewing the last position but not sufficient to estimate the direction of its movement.

In this study, update operation is performed by the UpdateEntities() method of EntityManager class. This method checks all the objects stored in the software one by one and updates the positions if necessary.

For the movement (translation animation) of the displayed objects, the graphics component is refreshed repetitively. In XNA applications, an Update() method is readily provided for this purpose [7]. This method is used to perform the changes for the scene to be redrawn.

The following assignment was used to allow the graphics component to refresh itself repetitively:

```csharp
Application.Idle += delegate
{ Invalidate(); }
```

The graphics component will draw the scene continuously as long as there is no event to be processed. UpdateEntities() will calculate the positions of the objects in every refreshment according to their initial and new positional data. Linear interpolation is used to find the points between these two limits of movement.

Linear interpolation is implemented by using the Lerp(...) method of the MathHelper class provided by XNA. In order to calculate the final position of an object; initial position, new position and the amount of increment that will be used as a step size are considered.

The formula used to calculate the final position by linear interpolation is as follows:

\[
\text{calculatedPosition} = \text{initialPosition} + (\text{newPosition} - \text{initialPosition}) \times \text{increment}
\]

The increment value is between 0 and 1. When the increment is 0, formula gives initial position and when increment is 1, formula gives the new position. The code segment implementation of the calculated position is given as:

```csharp
MathHelper.Lerp(position.X, newPosition.X, 0.01f);
```

This approach yields a gradual animated motion of military units in the scenario.

B. Military unit data storage

The developed Sandbox simulation software has facilitated the storing of the military unit data. This will allow the users to save some specific situations in the database and later query the objects according to some criteria such as object identification. Hence the data stored, in a way, will act as a military intelligence for future considerations. Microsoft SQL Server 2005 was used to design and develop the relational database [8].

1) Database design

The design of the database was planned according to a scenario concept that is embedded in the software. Each military situation is viewed as a new scenario and stored in the data base. This can be briefly explained as follows:

A military situation consists of a battlefield and the military units in it. These military units have to store enough information about a real world event using real world data in order to perform the visualization. Therefore the database design followed the same notation with the military symbology mentioned earlier.

Fig. 4 Diagram of relations between the tables in database

A specific military situation is stored in the Scenario table with a) its unique identification given with the DateTime table column and b) the battle field terrain table column as shown in Fig. 5. Entity is the table which stores the information of a military unit in data base.

The table used the military symbology with their fields such as Affiliation, Type, and Quantity etc. Report times and positions of military units were stored in the ReportDateTime and Position tables respectively. The mapping between a scenario and the entities given in that scenario was stored in the ScenarioEntity table.

2) Entity query

Access to the database was implemented using the ADO.NET API technology which allows accessing database programmatically. A separate form was designed to view the stored entities. This form enables users to access the military data directly and update it.

The approach followed in the software is also known as data binding [11] where the data in the database is accessed through the user interface components. A DataGridView was used to view the data retrieved from the database in a table. This control requires a DataSource object from which the data will be obtained in order to be displayed. Several queries were generated with these data sources in the implementation. The mapping between the data table and the data sources was...
performed by using TableAdapter class in order to execute the queries on the data source and return the results to the data table.

Three TableAdapters were used to provide access to Entity, Position, ReportDateTime tables. Each adapter has one or more queries to execute. A sample query used in the software was shown below:

```
SELECT ReportDateTime.EntityID,
       ReportDateTime.DateTime, Entity.Type,
       Entity.StaffComments
FROM ReportDateTime, Entity
WHERE(ReportDateTime.EntityID = @entityID AND Entity.ID =@entityID)
```

In the given SQL statement, data from ReportDateTime and Entity tables are retrieved as the query result. ID is the primary key in the Entity table and is a foreign key in ReportDateTime table. Therefore it was checked against the given entityID variable. The ‘@’ symbol denotes that entityID is a variable will receive its value in the runtime from the corresponding method of the table adapter.

III. RESULTS

Developed software allows users to open a USGS DEM terrain file and displays the terrain from different view-points such as vertical and orthographical views. The battlefield can be displayed using one of grass, sand or snow textures in order to model the terrain conditions in the battlefield.

Military units can be added to the battlefield by using the “Entity System” menu. This menu provides several features for addition, update or deletion of these units from the software.

The objects can be shown using both the symbology mentioned and the 3d models of military units. Users are able to select any unit from the graphical part and display its data on the left panel. This panel is also used to update the selected unit’s data. Changes are reflected on the graphical part in real time. Especially when the user updates the position information for an object, the object moves to the new position in an acceptably realistic manner.

A sample output of the software is given in Fig. 6 where the units are positioned in the battlefield. A tank battalion is located on the north-east and an enemy aircraft is approaching towards them. Artilleries are positioned on the east and a reconnaissance jeep is gathering field information on the west.

A different war scenario is given in Fig. 7. An air-defense unit is targeting a combat supply plane and a helicopter is delivering direct supporting fire to this unit. Infantries in the battlefield are simply shown by the symbology (boxes with cross on them).

A database was introduced in the developed software in order to store the data about military units. Additionally, scenarios are also saved in XML data format as proposed in [12].

Fig. 8 shows the data view part of the software which allows direct access to the database within the application. Users are able to make several queries about a specific unit. Results of the queries are displayed in a data table.
IV. CONCLUSION

Sandbox will provide commanders with recent and reliable data in order to improve military decision making processes. A realistic model of the battlefield will generate a better perception of the current situation and develop winning strategies.

Software design was kept simple. However it has a comprehensive coverage of the necessary military information and it is also flexible enough to respond to future requirements.

3D models additionally presented in this study with military symbology as opposed to implementations in [1], [4], [13] and [14] allowed users to differentiate between several symbols more efficiently. This facilitated a better and easier military training for the army. Time concept was not included in [9], [10]. The developed Sandbox introduced the time data in the form of time tags added to the position reports of the military units to generate position versus time archive.

The work is can be extended by introducing the web services for remote users to change the state of the battlefield while presently remote users can only view the current state of the battlefield.

Finally, Sandbox will bring new initiatives for battlefield management and for simulation of the war games by using both the visual features and the advanced data handling capabilities of the developed software. The software can be accessed freely from http://comp.eng.ankara.edu.tr

REFERENCES