SSstory: 3D data storytelling based on SuperSQL and Unity

Jingrui Li
Dept. of Information and Computer Science,
Keio University
Yokohama, Japan
li@db.ics.keio.ac.jp

Kento Goto
Dept. of Information and Computer Science,
Keio University
Yokohama, Japan
goto@db.ics.keio.ac.jp

Motomichi Toyama
Dept. of Information and Computer Science,
Keio University
Yokohama, Japan
toyama@ics.keio.ac.jp

ABSTRACT

SuperSQL is an extended SQL language, which brings out a rich layout presentation of a relational database with a particular query. This paper proposes SSstory, a storytelling system in a 3D data space created by a relational database. SSstory uses SuperSQL and Unity to generate a data video and add cinematic directions to the data video. Without learning special authoring tooling, users can easily create data videos with a small quantity of code.

CCS CONCEPTS

• **Human-centered computing** → *Visualization systems and tools.*

KEYWORDS

SuperSQL, databases, data visualization, data storytelling

ACM Reference Format:
Jingrui Li, Kento Goto, and Motomichi Toyama. 2021. SSstory: 3D data storytelling based on SuperSQL and Unity. In *25th International Database Engineering & Applications Symposium (IDEAS 2021), July 14–16, 2021, Montreal, QC, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3472163.3472277

1 INTRODUCTION

Nowadays, with a mass of data created, how to tell the insights of data to the audience is becoming critical. Therefore the data storytelling approach to telling the data insight as a story becomes a topic in data visualization.

Previous research[1] named SSQL4.5DVS uses Unity[2], a game development platform, to implement a system that can generate objects with data in a 3D space. With SSQL4.5DVS, even users without knowledge of Unity can generate visualized scenes in the 3D space.

However, SSQL4.5DVS is just a visualization tool which not support the function of data storytelling. Also, the authoring tool for the data storytelling existing but such tool takes time for learning or needs specialized knowledge and technology.

This paper proposes data storytelling systems that use SuperSQL, an extended SQL language, and Unity to make a data video in the 3D space.

IDEAS 2021, July 14–16, 2021, Montreal, QC, Canada

ACM ISBN 978-1-4503-8991-4/21/07…$15.00
https://doi.org/10.1145/3472163.3472277

2 RELATED WORK

Several papers inspired our paper. Edward et al.[3] looks at the construction of data storytelling and analyzes the element. This paper refers to the structure of data storytelling. Fereshteh Amini et al.[4] analyze 50 of professionally designed data videos, extracting and exposing their most salient constituents. Also, Fereshteh Amini et al.[5] develops a tool named DataClips which can generate data video with the data clips. Users can easily do it without using a video authoring tool like Adobe AfterEffects. Ren[8] looks at the importance of annotation in data storytelling and develops a specialized data storytelling tool for the annotation. Edward et al.[6] research the element of stories then sum up the design pattern of the data story with the result. Lyu et al.[7] shows the communication model of storytelling and introduces a visualization works, a Chinese painting, using the big meteorological data in China. Amin Beheshti et al.[9] combines the intelligent DataLake technique and data analytics algorithm to implement an interactive storytelling dashboard for social data understanding. Shi et al.[10] develop a tool that automatically generates data stories from Google Spreadsheet.

However, these related works expend time costs to generate data stories. Also, most of them present the data story as a 2D graph; it is hardly any approach for data storytelling in 3D space.

3 SSstory

This paper proposes an approach that combines the StoryGenerator and the StoryEditor. StoryGenerator uses SuperSQl and Unity to visualize the data from the database, and the StoryEditor adds the story elements to the work of visualization. The presentation of the data story will be a data video, which means show the data as a video or animation.

3.1 High-dimensional visualization object

A High-dimensional visualization object can show high-dimension information. For example, an object like a human face can allocate the height of the nose, the width of the mouth, the color of the eye to high-dimension information.

We can create a High-dimensional visualization world with several High-dimensional visualization objects. The advantage of the High-dimensional visualization object and High-dimensional visualization is that the audiences can choose the information they want to know.

Figure 1 shows a result that uses the asset of Unity to create a world just like a farm[14]. In this world, the strength of wind, the strength of sunlight, the number of animals, the amount of grass can have information from the database.

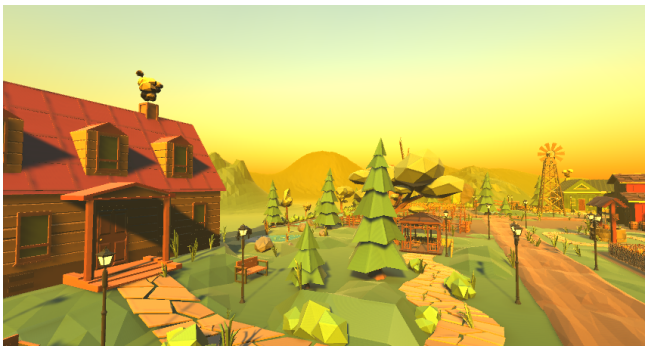


Figure 1: Asset of Unity

3.2 About SuperSQL

SuperSQL is an extended language of SQL that structures the output result of the relational database, enables various layout expressions, and has been developed at Keio University Toyama Laboratory[11][12]. The query replaces the SQL SELECT clause with a GENERATE clause with the syntax of GENER-ATE <media> <TFE>, where<media>is the output medium and can specify HTML, PDF, etc. Also, <TFE> represents a Target Form Expression that is an extension of the target list and is a kind of expression having a layout specifying operators such as a connector and an iterator.

3.3 Architecture

Figure 2 shows the architecture of the SSstory. The system consists of the StoryGenerator and the StoryEditor. The StoryGenerator consists of the Parser that distinguishes the SQL query and layout presentation, the DataConstructor that layout the data from the result of the database and structured information of the table, the CodeGenerator to generate XML file according to the structured data. In the StoryEditor, the Timeline and Cinemachine package is used to improve the quality of DataVideo. This system aims to generate 80% of the material by StoryGenerator and finish the video with the rest 20% adjustment by StoryEditor.

Then we will show the workflow of this system. First, store the data user wants to visualize in the database. When the SuperSQL query is executed, the XML file and C# file are generated based on the data and the query contents. The information of the object and the elements given on the object and the layout are described in the XML file. C# file reads its XML file and creates objects, gives color and animation for objects, and determines the position of each object from layout information. By importing these two script files into Unity and executing it, we can realize the data visualization specified in the query. In the case of using assets, we also need to import assets to Unity at the same time. Then in the StoryEditor, the Cinemachine and AnimatorController will create the animation element through the XML file. Then the user can use the Timeline to choose the activated camera and adjust the video sequence. Figure3 shows the Timeline used by StoryEditor.

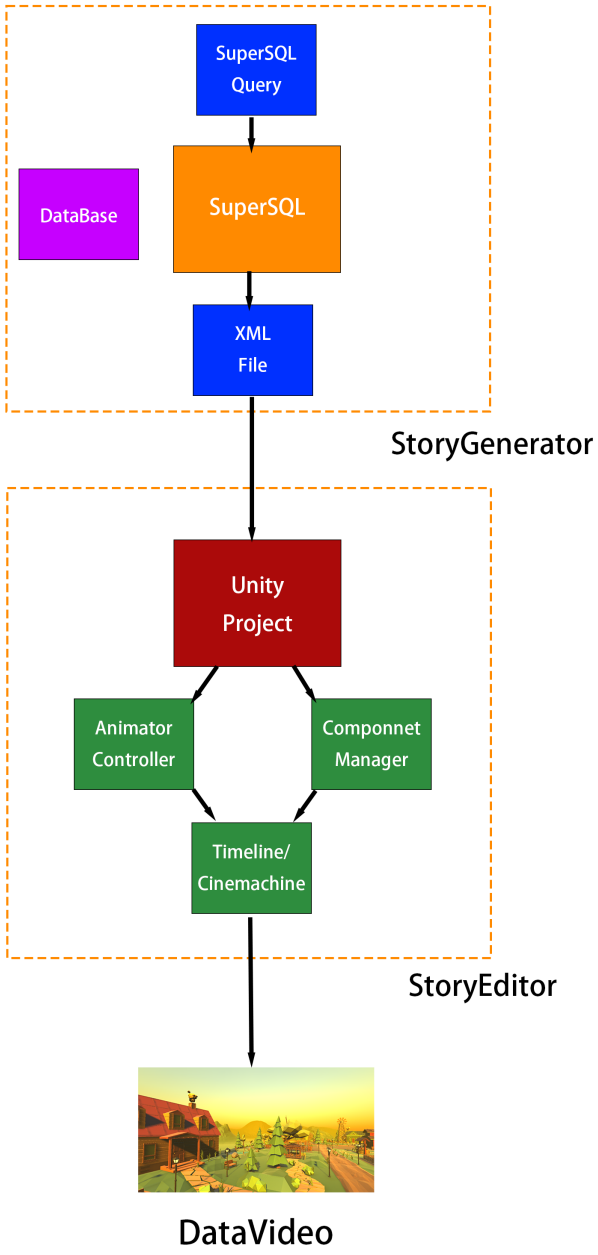


Figure 2: Architecture of SuperSQL

3.4 Object Generation Function

This system uses the asset function to create objects which property likes size can be assigned with the argument. The system supports three types of object generation functions.

3.4.1 Primitive Generation Function. User can create primitive object like cube, sphere, torus, cuboid, pyramid by using this function.

$$object(object, argument1, argument2)$$

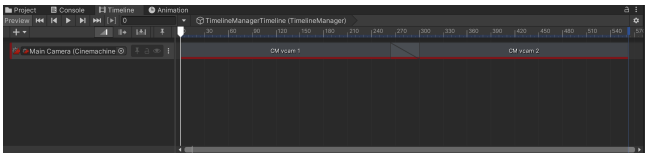


Figure 3: Timeline of Unity

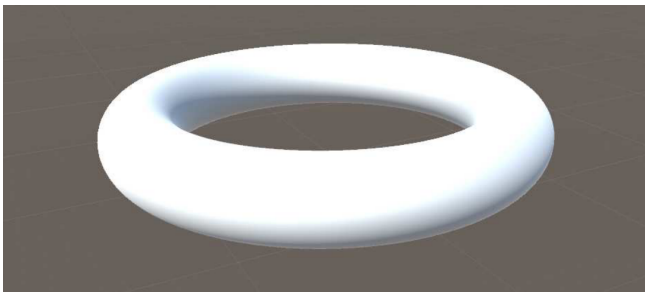


Figure 4: Generation example of object(torus)

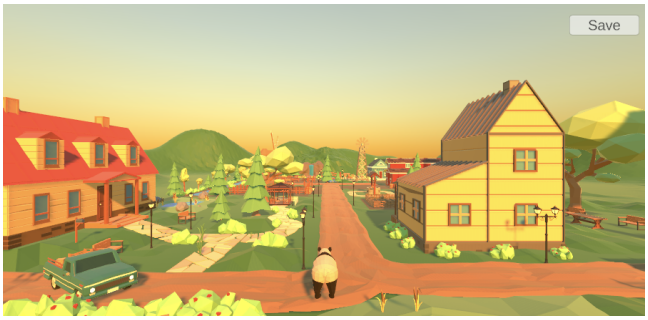


Figure 5: Generation example of asset(panda)

The following shows a description example of the object function for generating an object (torus) and its generation result.

$$object('torus', r1, r2)$$

Also the 3D text object could be generated though the primitive generation function

$$object('text', text, size)$$

3.4.2 Asset Generation Function. Asset generation function uses the asset, an existed object created by the user.

$$asset(asset\_name, size)$$

Figure5 user can generate an asset called Pandan through the following description.

$$asset('Panda', size)$$

3.4.3 Random Generation Function. The random generation function will generate objects from a folder. All the assets in the folder will be put in an array and wait to be generated. By default, assets will be generated in the X-Y plane with normal distribution, and the max coordinate is according to the scene asset.

$$random(asset\_folder, number, min\_size, max\_size)$$

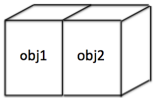
For example, this query can generate all assets in the folder of Animal according to the argument. If the user wants to generate them in the whole X-Y-Z space, not the X-Y plane, the user can write a decoration(@) for the function.

$$random('Animal', number, min\_size, max\_size)@\{space = 'X - Y - Z'\}$$

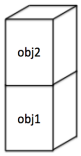
3.5 Connector

Table 1 shows the operator supported by the SuperSQL. The connector is an operator that specifies in which direction (dimension) the data obtained from the database is to be combined. There are the following three kinds. The character used to represent each connector in a SuperSQL query is noted in parentheses.

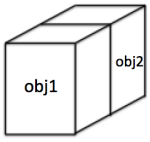
- Horizontal Connector(,).
  Two pieces of data connected by this connector are arranged next to each other horizontally (along with the x-axis). In a flat document, this means the two pieces of data are shown on the same line. In a 3D document, this connector acts as shown below.
  ex: obj1, obj2



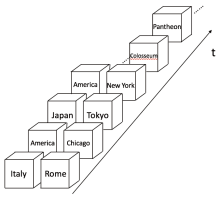
- Vertical Connector(!).
  Two pieces of data connected by this connector are arranged next to each other vertically (along the y-axis). In a flat document, this means the two pieces of data are shown on two separate lines. In a 3D document, this connector acts as shown below.
  ex: obj1! obj2



- Depth Connector(%).
  Two pieces of data connected by this connector are arranged next to each other in-depth (along with the z-axis). This means the first piece of data links to the second piece of data in a flat document. In a 3D document, this connector acts as shown below.
  ex: obj1% obj2



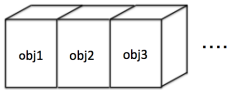
- Time Connector(#).
  Two pieces of data connected by this connector are arranged next to each other in the time axis. It will be shown at a constant time interval. Following description shows an example for Time Connector.
  ex : [country, city]##[building]#



3.6 Grouper

The second type of operator available in SuperSQL is the grouper. When a grouper is used, the attribute or group of attributes affected by the operator is repeated in the document for each tuple retrieved from the database by the query. The following groupers are available in this system.

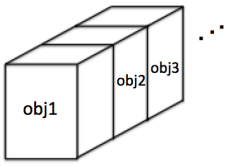
- Horizontal Grouper([ ],).
  An object or group of objects is added to the document for each tuple in the relation retrieved by the query, each object is arranged horizontally. In our 3D system this produces the pattern shown below.
  ex : [Obj],

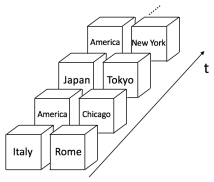


- Vertical Grouper([ ]!).

  An object or group of objects is added to the document for each tuple in the relation retrieved by the query, each object is arranged vertically. In our 3D system this produces the pattern shown below.
  ex : [Obj]!

- Depth Grouper([ ]%).
  An object or group of objects is added to the document for each tuple in the relation retrieved by the query; each object is arranged in depth. In our 3D system, this produces the pattern shown below.
  ex : [Obj]%
- Timeline Grouper([ ]#).
  The timeline grouper will arrange scenes to a timeline and present the data as a data video.
  ex : [country, city]#





- Slideshow Grouper([ ]&).
  Slideshow grouper will arrange scenes like a slideshow with a UI button. Click the button can translate to another scene. Chapter 4.3 will show a example for slideshow repeat.

Table 1: Operator in SuperSQL

tfe1, tfe2	Horizontal connector
tfe1! tfe2	Vertical connector
tfe1% tfe2	Depth connector
$[tfe]$,	Horizontal grouper
$[tfe]$!	Vertical grouper
$[tfe]$%	Depth grouper
$[tfe]$#	Timeline grouper

3.7 Environmental Function

As a result of the environmental function, the information of the environmental object will be assigned. The environmental object is a kind of object that unique exist in the scene. For example, there is only one sun in the scene, and it will be an environmental object. Through the following query, the user can vary the strength of sunlight along the timeline.

$$env(object, component, attribute, value)$$

When the environmental function is executed, an empty GameObject called EnvController will be generated in Unity, and the script of the EnvController will fetch the environment object and its component, assign a value to the attribute.

$$env('Sunlight','Light','intensity', value)$$

This function can assign the value of for the Sunlight object to change the lighting in the scene.

3.8 2D Element Function

This system supports two types of 2D element functions. 2D element means an object that exists in the canvas. Any connector will not influence its coordinate so that it could be written in any place in the code. The 2D element also could be repeated by a timeline grouper to generate an animation.

Figure 6: Example of annotation

3.8.1 Annotation Function. An annotation function will be used to create an annotation for canvas.

$$annotation(type, text)$$

The first argument shows the type of annotation. There are some presets for the annotation like "text_bottom," which means a text box at the bottom of the canvas.

$$annotation('text\_bottom','some\ annotations')$$

Figure6 shows the result of this query. Also, you can use decoration to complete the detail of annotation such as width, height, font size, texture, and highlight.

$$annotation('text\_bottom','some\ annotations')@\{font\_size =' 20'\}$$

Fig. will show the result of this query and its decoration. Another valuable decoration for the annotation function is setting the start time to achieve a better presentation for the data video.

$$annotation('text\_bottom','some\ annotations')@\{start\_time =' 5'\}$$

The annotation will appear after 5 seconds, according to the decoration. Fig shows a combination of three annotation functions.

$$[annotation('text\_bottom', data)]\#$$

The timeline grouper with the annotation function can make the annotation to be an animation annotation to show the data.

3.8.2 Chart Function. Chart function will be used to create charts in the canvas. The first argument shows the type of chart. Some presets of charts have been prepared. The data of the chart is assigned by the second and the third argument.

$$chart(type, data1, data2)$$

We use the asset of Unity called 'Graph and Chart'[13] to implement this function. The table shows the supported type of chart in the chart function. Figure7 shows a sample of Graph And Chart asset.

$$[chart(type, data1, data2)], [chart(type, data1, data2)]!$$

Both horizontal grouper([],) and vertical grouper([]!) will repeat the data and make it be a whole chart. The graph shows the grouper used for the chart function.

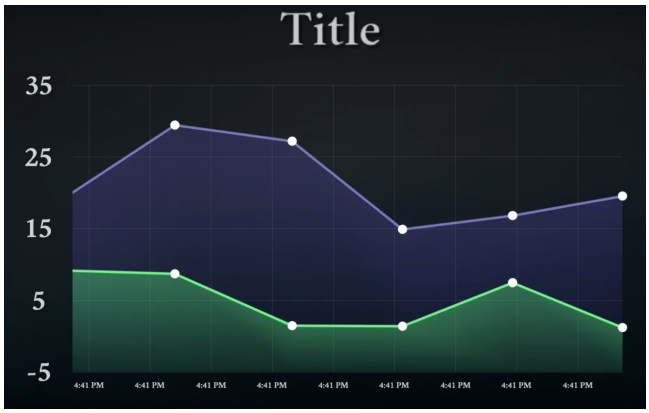


Figure 7: Graph And Chart

$$[chart(type, data1, data2)]\#$$

The timeline grouper can make the chart to be an animation chart.

Like the annotation function, you can use the decoration to adjust the chart, like width, height, axis label, and approximate curve.

$$chart(type, data1, data2)@\{approximate\_curve =' linear'\}$$

Two charts can be combined as one chart by using the mark of '+'.

$$chart('Bar', data1, data2) + chart('LineGraph', data3, data4)$$

3.9 Optional function

The optional function is a function used to add an extra attribute to the object generation function. This section will introduce the optional function and show some examples.

$$function(< TFE >, parameter)$$

The first argument <TFE> means the object generation function with its connector and grouper. The second argument shows the extra attribute given to the TFE.

- Hop Function
  The hop function can cause an object to jump by describing velocity and vertex and axis direction. The following shows a description example of hop function and its image.
  $$hop(< TFE >, velocity, vertex, axis)$$
- Pulse Function
  The pulse function can stretch the object by describing magnification and velocity. The following shows a description example of pulse function and its image.
  $$pulse(< TFE >, magnification, velocity)$$
- Rotate Function
  The pulse function can rotate the object by describing the rotation rate for three axes. The following shows a description example of pulse function and its image.
  $$rotate(< TFE >, X\_rate, Y\_rate, Z\_rate)$$

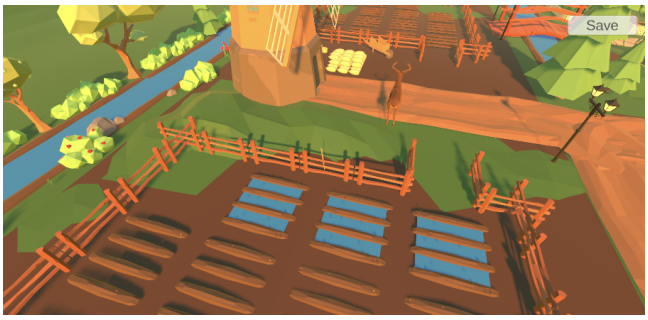


Figure 8: Example of follow camera

- Color Function
  The color function gives an object the color by describing with a character string. Available colors include "red", "blue", "green", "black", "clear", "cyan", "gray", "yellow", "white", "magenta". The following shows a description example of color function.

$$color(< TFE >, color\_name)$$

- Position Function
  The position function is a valuable function to change the absolute coordinate for the object. It original point(0, 0, 0) of 3D space as a center and update the coordinate after generation.

$$position(< TFE >, x\_coordinate, y\_coordinate, z\_coordinate)$$

- Move Function
  The move function can change the relative coordinate for the object. The move function is different from other functions. If a move function is described in a move function again, the coordinate will be calculated through all the move functions.

$$move(< TFE >, x\_coordinate, y\_coordinate, z\_coordinate)$$

- Optional Annotation Function
  The optional annotation function is a function to assign annotations to an object. Unlike the annotation function in the section, which generates annotation in the canvas, the optional annotation function adds some annotations to an object. The user can get the annotation by clicking the object in the game mode.

$$optional\_annotation(< TFE >, text)$$

- Camera Function
  Camera and cinema direction play an important role in data storytelling. This system supports the camera function to generate a camera in the scene. The type of camera including follow camera, a camera that follows an object, a fixed camera, a camera in a stable position, and FPS camera, a camera to show the first-person view of an object.

$$camera(< TFE >, type)$$

  Figure8 shows a follow camera for a deer asset.

$$camera(asset('Deer', 1), 'Follow')$$

  Moreover, the camera function can also be repeated by grouper to generator a camera for each object. Decoration also can be used to change the attribute of the camera, such as lens, damping, and coordinate.

3.10 Query

In general, the query of storytelling will be the following expression.

```
GENERATE unity_dv
Scene(scene_name)
[option to object(
several functions(...), necessary parameter)(
    grouper)(connector)
(environmental function)
(2D element)
FROM table
WHERE condition
```

(First element)Describe the scene used in this system.

(Second element)SSstory supports several functions to assign the information to objects. Decide which element assign to the objects through the function name, then describe the object generate function or other function in the first argument. It is necessary to describe the argument in the correct order. The connector will be used to connect another object here. Each of the objects will be the same expression rule.

(Third element)Assign information to environmental objects through the environmental function. There is only one environmental function for an environmental object in a scene.

(Fourth line)Describe 2D elements in the canvas. Other 3D objects would not influence their coordinate in the second element.

The first and second element is must, and the third and fourth element is optional.

3.11 Implementation

This section will show the implementation of StoryGenerator.

3.11.1 XML File Generation. The XML file will be created by the SuperSQL through the "*.ssql" file.

Then we will explain Algorithm1. First, input the tree structure data, layout formula through the query in the '*.ssql' file. If the CodeGenerator in the SuperSQL reads the media name of Unity_dv, it will add an XML tag to the output. Then, SuperSQL will read the grouper and generate the grouper node with the attribute of g1(,), g2(!), g3(%), g4(#), g5(&). Next time, read the child node of the grouper, generate the connector node with the attribute of c1(,), c2(!), c3(%), c4(#). The object around the connector will be a child of the connector node.

Then read the optional function of the query. If the function is an object generation function, creating the object or asset node. Then read the optional function, add the information to the object or asset node until the last function.

3.12 Object Generation

The Algorithm2 shows how the object is created in Unity. First, the XML reader will read the XML file for all elements. If there is a grouper element, generate the grouper object, a wrapper for the child object, and read its child element. Then if there is a connector element, generate the connector object, a wrapper for the child object, and read its child element. When reading an object

Algorithm 1 XML File Generation

Require: tree structure data, layout formula
Ensure: XML file
add XML tag
for all operator **do**
  if there is the grouper **then**
    create grouper element
  end if
  if there is the connector **then**
    create connector element
  end if
  while the function has function in argument **do**
    save information of current function
    read next function
    if object generation function **then**
      create object element
      add saved optional function information as child node of object
    end if
  end while
end for
return XML file

Algorithm 2 Object Generation

Require: XML file
Ensure: Object with function information
read XML file
for all XML element **do**
  if there is the grouper element **then**
    create grouper object
  end if
  if there is the connector **then**
    create connector object
  end if
  while the element has child **do**
    if object generation function **then**
      create object
    end if
    if optional function **then**
      add optional function information to the object
    end if
    if environment function **then**
      generate environment manager
      find the environment object and assign value
    end if
  end while
end for
return Object

generation element, instantiate an object by the information of the object element. Then if the object element has an optional function as a child, add the information to the object until the last child.

4 USE CASE

This chapter shows the use case of storytelling with Unity and SuperSQL.

4.1 Sample Data

The use case uses the weather data about nine dimensions of Tokyo 2020[16] and an asset of farm[14]. Table 2 shows the corresponding relationship between data information and asset property. The environment of the scene is decided through the following description of an environmental function, and Table 3 shows the several environmental functions supported by the system.

Table 2: Corresponding relationship of information and asset

Data information	Asset property
air pressure	strength of lighting
max precipitation	size of grass
average precipitation	amount of grass
average temperature	number of animals
min temperature	min size of animals
max temperature	max size of animals
humidity	velocity of the waterfall
wind speed	rotate speed of the windmall
time of sunlight	number of birds (higher air pressure more bird)

Table 3: Environmental Function

Function	Asset property
env('waterfall', 'rotate' , 'velocity', value)	velocity of waterfall
env('windmill', 'rotate', 'velocity', value)	velocity of windmill
env('sunlight', 'light', 'intensity', value)	strength of sunlight
env('grass', 'transform', 'scale', value)	size of grass

4.2 Query Example

The following query is executed, and the chapter shows the result of it. -QueryExample

```
GENERATE Unity_dv
 scene(farm),
 [random('animal', w.tempera_average, w.
     tempera_min, w.tempera_max),
    assert('bird', w.pressure),
    optional_annotation(env('waterfall', '
        ParticleSystem', 'speed', w.humidity),
        'The humidity of' || w.month || 'is' ||
         w.humidity),
    optional_annotation(env('Sun','Light', '
        intensity', w.sunlight_time), 'The
        sunlight time of' || w.month || 'is' ||
         w.sunlight_time),
```

```
    optional_annotation(env('grass', '
        ParticleSystem', 'max_size', w.
        precip_max), 'The max of precipitation
        of' || w.month || 'is' || precip_max),
    optional_annotation(env('grass', '
        ParticleSystem', 'max_particles', w.
        precip_total), 'The total of
        precipitation of' || w.month || 'is' ||
         w.precip_total),
    optional_annotation(env('wind', 'rotate', '
        velocity', w.wind), 'The strength of
        Wind of' || w.month || 'is' || w.wind),
    w.month
]&

from weather_tokyo w
```

The scheme used in this query shows the following.
-weather(id, pressure, precip_max, precip_total, tempera_average, tempera_min, tempera_max, humidity, wind, sunlight_time);

4.3 Execution Result

This chapter shows the result of execution. Figure 9 shows the dataset of August, and Figure 10 the dataset of December. You can see that animal of August is bigger than in December; also, the sunlight is more bright.

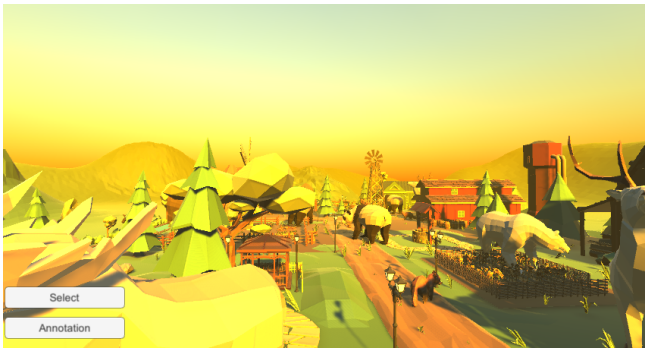


Figure 9: Dataset of August

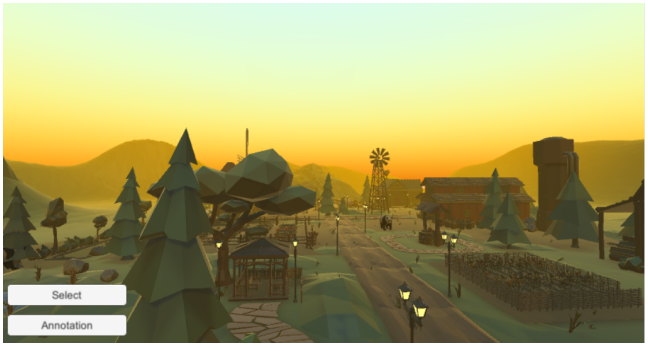


Figure 10: Dataset of December

Figure 11: Example of slideshow repeat

- Annotation
  Annotation can be assigned to an object through the optional annotation function shows in Figure 13.
- Slideshow repeat.
  Figure 11 shows the example of the slideshow repeat. Grouping with the month, then scenes and corresponding UI buttons of the month will be generated.

Figure 12: Annotation for scene

Figure 13: Annotation for object

4.4 Another Presention

SuperSQl can alter the presentation easily by just rewrite the query. For this use case, we chance the corresponding relationships of air pressure and sunlight time. Higher air pressure will show a more bright light in a new presentation, and longer sunlight time will bring more birds.

Figure 14: Dataset of August after

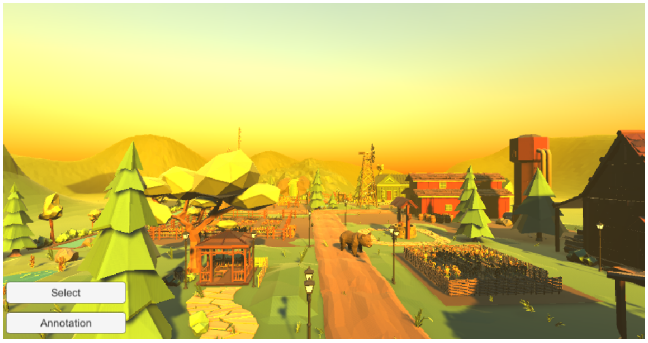


Figure 15: Dataset of December after

Figure 14 and 15 shows the new presentation of the same dataset. It is known that December is more bright than August because of the higher air pressure. In SSstory, creators can improve their presentation quickly to tell the message expectedly.

4.5 Another Dataset

SSstory can use the same asset to present other datasets. As an alternative, we use the data of C-VOID19 of 2020 in Tokyo, Japan. Table4 shows the corresponding relationship in a new dataset.

-QueryExample2

```
Generate Unity_dv
scene('Farm')
    ,random('Bird', c.total_vaccine/100000,
        0.1, c.newly_vaccine/500000 +0.5)
    ,env('Directional Light', 'Light', '
        intensity', (8000 - c.newly_infection)
        /3000)
    ,env('GrassA', 'ParticleSystem', 'startSize
        ', c.newly_death/2)
    ,env('GrassA', 'ParticleSystem', '
        maxParticles', c.total_severe*2)
    !annotation('textbottom', c.date || ',' ||
        'newly infection' || ':' || c.
        newly_infection)

from covid19 c
where c.date = '2021-3-22'
```

Table 4: Corresponding relationship in Covid-19 dataset

Data information	Asset property
newly infected cases	strength of lighting (less people more bright)
total severe cases	size of grass
newly death cases	amount of grass
total vaccinations	number of birds
newly vaccinations	max size of birds

Figure 16: Dataset of 22nd March 2021 about Covid-19

Figure 17: Dataset of 22nd April 2021 about Covid-19

Figure 18: Dataset of 22nd May 2021 about Covid-19

Figure 16 and 17 show the result of visualization. New infected, severe cases, death cases were increased from March to April 2021. Also, the vaccinations were increased through the number of birds. The result of 22nd May(Figure 18) has lush grass also the increasing birds which show the infection condition intuitively. From this visualization work, we can tell a message that even though the virus spread is uncontrolled, the inoculation was increased, and its effect will be excepted.

4.6 Another Asset

In this section, we will show an alternative asset for the same dataset of C-void19. Table5 shows the corresponding relationship in the Covid-19 dataset with a new asset, an amusement park[15].

Table 5: Corresponding relationship in Covid-19 dataset with a new asset

Data information	Asset property
newly infected cases	strength of lighting (less people more bright)
total severe cases	size of foliage
death cases	amount of foliage
total vaccinations	size of attractions
newly vaccinations	rotate rate of attractions

-QueryExample3

```
Generate Unity_dv
scene('Park')
    ,env('Bushes', 'transform', 'scale_child',
        c.total_severe/400)
    ,env('Trees', 'transform', 'scale_child', c
        .newly_death/50)
    ,env('Directional Light', 'Light', '
        intensity', (8000 - c.newly_infection)
        /5000)
    ,env('FerrisWheel_Rotate', '
        FerrisWheel_Rotation', 'speed', c.c.
        newly_vaccine/1000)
    ,env('Carousel_Rotate', 'Carousel_Rotation
        ', 'speed', c.c.newly_vaccine/1000)
    ,env('FerrisWheel', 'transform', 'scale', c
        .total_vaccine/7000000 + 0.5)
    ,env('Carousel', 'transform', 'scale', c.
        total_vaccine/7000000+ 0.5)
    ,annotation('textbottom', c.date || ',' ||
        'newly infection' || ':' || c.
        newly_infection)

from covid19 c
where c.date = '2021-3-22'
```

Figure 19: Dataset of 22nd March 2021 about Covid-19 with a new asset

Figure 20: Dataset of 22nd April 2021 about Covid-19 with a new asset

Figure 21: Dataset of 22nd May 2021 about Covid-19 with a new asset

Figure19, 20, and 21 show the infection condition of 22nd March, Apil, and May. We show the range of presentation of this system from these visualization works because users can choose any asset they want to use.

5 EVALUATION

For evaluation, we get ready to invite the experimenter to use this system and compare it with other data storytelling systems in expression, learning cost, and amount of code.

6 CONCLUSIONS

We implement the StoryGenerator to generate a static data video through the timeline grouper at the present stage. We will use the cinematic asset to design and implement the StoryEditor to achieve a high-quality and more ac data video in the future.

REFERENCES

[1] Tatsuki Fujimoto, "3D Visualization of data using SuperSQL and Unity", IDEAS, 2019.
[2] Unity. https://unity3d.com/.
[3] Edward Segel, Jeffrey Heer, "Narrative Visualization: Telling Stories with Data", IEEE Transactions on Visualization and Computer Graphics, 2010.
[4] Fereshteh Amini1, Nathalie Henry Riche, Bongshin Lee, "Understanding Data Videos: Looking at Narrative Visualization through the Cinematography Lens", HAL, 2015
[5] Fereshteh Amini, Nathalie Henry Riche, "Authoring Data-Driven Videos with DataClips", IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, 2017.
[6] Edward Segel, "Narrative Design Patterns for Data-Driven Storytelling", IEEE Transactions on Visualization and Computer Graphics, 2010.
[7] Yanru Lyu, "Visual Data Storytelling: A Case Study of Turning Big Data into Chinese Painting", HCII 2020.
[8] Donghao Ren, "ChartAccent: Annotation for Data-Driven Storytelling" , IEEE Pacific Visualization Symposium, 2017.
[9] Amin Beheshti, Alireza Tabebordbar, Boualem Benatallah, "iStory: Intelligent Storytelling with Social Data", WWW '20 Companion, 2020
[10] Danqing Shi, "Calliope: Automatic Visual Data Story Generation from a Spreadsheet", IEEE Transactions on Visualization and Computer Graphics, 2020.
[11] SuperSQL. http://ssql.db.ics.keio.ac.jp.
[12] M. Toyama, "Supersql: An extended SQL for database publishing and presentation", Proceedings of ACM SIGMOD '98 International Conference on Management of Data, pp. 584–586, 1998.
[13] BitSplash Interactive, Chart And Graph, https://assetstore.unity.com/packages/tools/gui/graph-and-chart-78488.
[14] Must Have Studio, HappyLifeville.Low Poly Farm, https://assetstore.unity.com/packages/ 3d/environments/industrial/happylifeville-low-poly-farm-163983
[15] Must Have Studio, Amuseville. Low Poly Park, https://assetstore.unity.com/packages /3d/environments/urban/amuseville-low-poly-park-192985
[16] Japan Meteorological Agency. https://www.data.jma.go.jp/