Slovak University of Technology in Bratislava
Faculty of Informatics and Information
Ilkovičova 2, 842 16 Bratislava 4

Team project

Engineering work

Team number: 12
Team name: 3DSpaceGen
Members: Ján Antal, Marek Drahoš, Matej Mikuš, Dominik Mazák, Peter Pápay, Andrej Pisarčík, Adam Poperník
Team leader: Ing. Karol Rástočný, PhD.
Ac. year: 2018/19

Content

1 Big picture

In this section is presented the project created by 3DSpaceGen team. The goals of the project are presented in this document. Document describes the whole architecture, but also goes into details of each module.

1.1 Global project goals

1.1.1 Winter term

Output of our project in winter term is an application programmed in Unity for HTC Vive device. This application will support dynamic insertion of objects into scene and also manipulation of those objects. The objects will be generated with neural networks, specifically using generative adversarial networks. Inference will work in real time after accepting a request from client through provided REST API.

1.1.2 Summer term

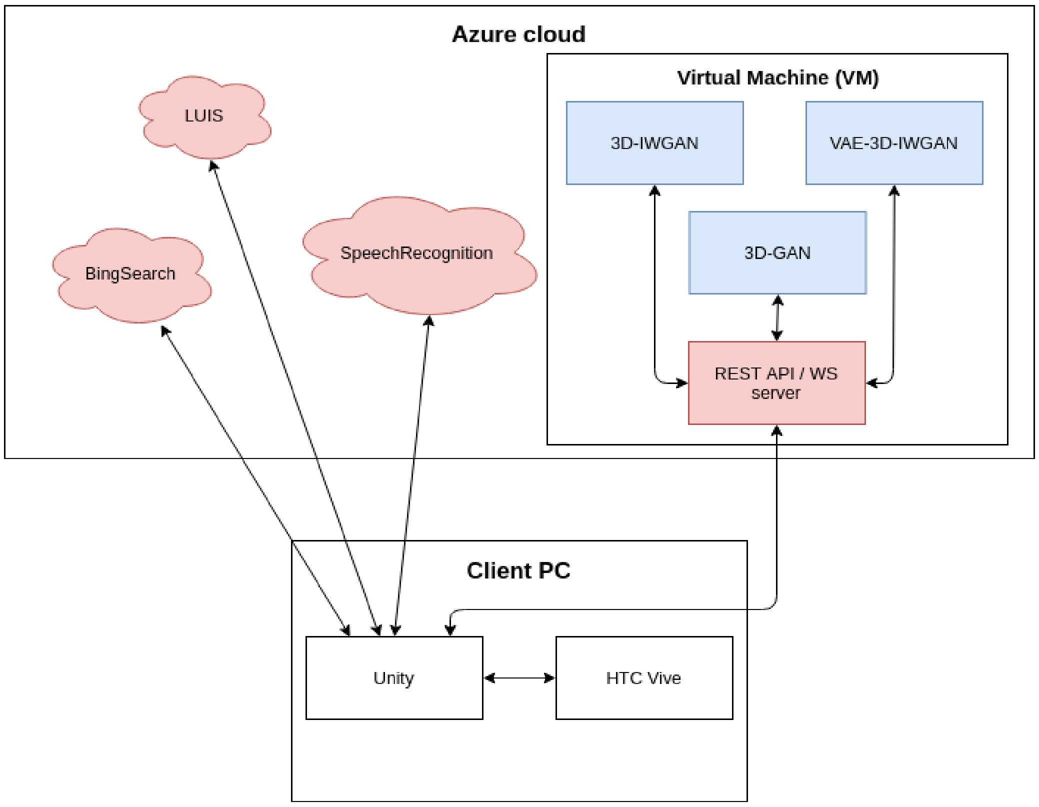
The goal of summer term is to improve existing application. Main goal is to implement a feature, that user can modify the structure of generated objects. Voice control of application will also be added, to increase the user experience.

1.2 Overall system architecture

We have identified, described, and implemented the following architecture. The client is a basic personal computer with Unity installed and HTC Vive connected. The user interacts with the world using the HTC Vive controllers. This client sends requests to REST API service, in order to perform methods with 3D models. REST API service uses remote inference clients in order to perform these methods.

Communication between the inference machine clients (IM) and virtual machine server (VM) is described below. VM represents a server role that handles user requests and routes them to specific IM according to request specification. VM communicates with Inference machines using the WebSocket, which can invoke the request for generating a specific 3D model. Inference machine is used to perform various methods with 3D models (pre-trained models of GAN). When SVM gets a result from IM, it sends the result back to a user in the response message.

New models are pushed into the repository by hand and trained on external computers. Trained models are then loaded from the shared git repository for every client.



Img. 1: System architecture.

2 Modules

2.1 Virtual reality module

The module allows the user to manipulate with 3D objects, change their position, size, rotation, also allows the model to be stored in the space according to the user's requirements. Through this module, the user can insert individual 3D objects into the scene.

Analysis

For the space visualization, we decided that the virtual reality will be more suitable for our solution unlike the augmented reality, because it allows to change the whole space and the user is not is not limited by the real objects in the scene.

Design

We suggest creating a simple menu to allow the user to select the category of the desired object which will be spawned in the scene.
When the object is spawned, the user will be able to manipulate the object (change the scale, position and rotation) via the HTC Vive controllers and delete it.

Parts:

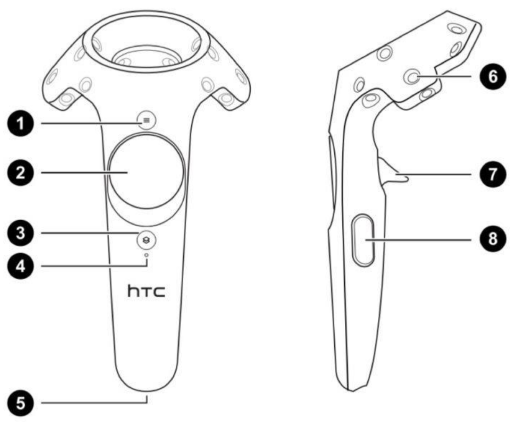
- **Unity 3D** is a multi-platform gaming engine developed by Unity Technologies. Unity provides development of 2D and 3D games also in virtual reality using SteamVR plugin. In addition to the graphical creation environment, it also supports the creation of scripts primarily in C #.

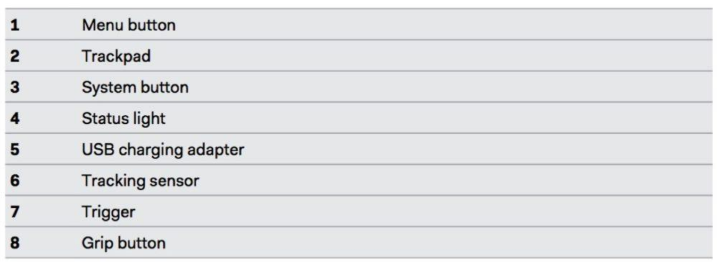
- **HTC Vive** is a virtual reality head-mounted display. It uses "room scale" tracking technology to allow the user to move in 3D virtual space and motion-tracked handheld controllers to interact with the environment.

- **VRTK** (Virtual Reality Toolkit) is a collection of useful scripts and concepts for building VR solutions rapidly and easily in Unity3D.
  We use implemented VRTK solutions such as:
  - Head-set menu with 2D buttons.
  - Interactions like touching, grabbing and using objects
  - Interacting with Unity3d UI elements through pointers or touch.

Manipulations with objects via controllers

The controller has several buttons that allow user to manipulate with the application in virtual reality. The user can manipulate/interact with objects via this controllers which are described in picture.

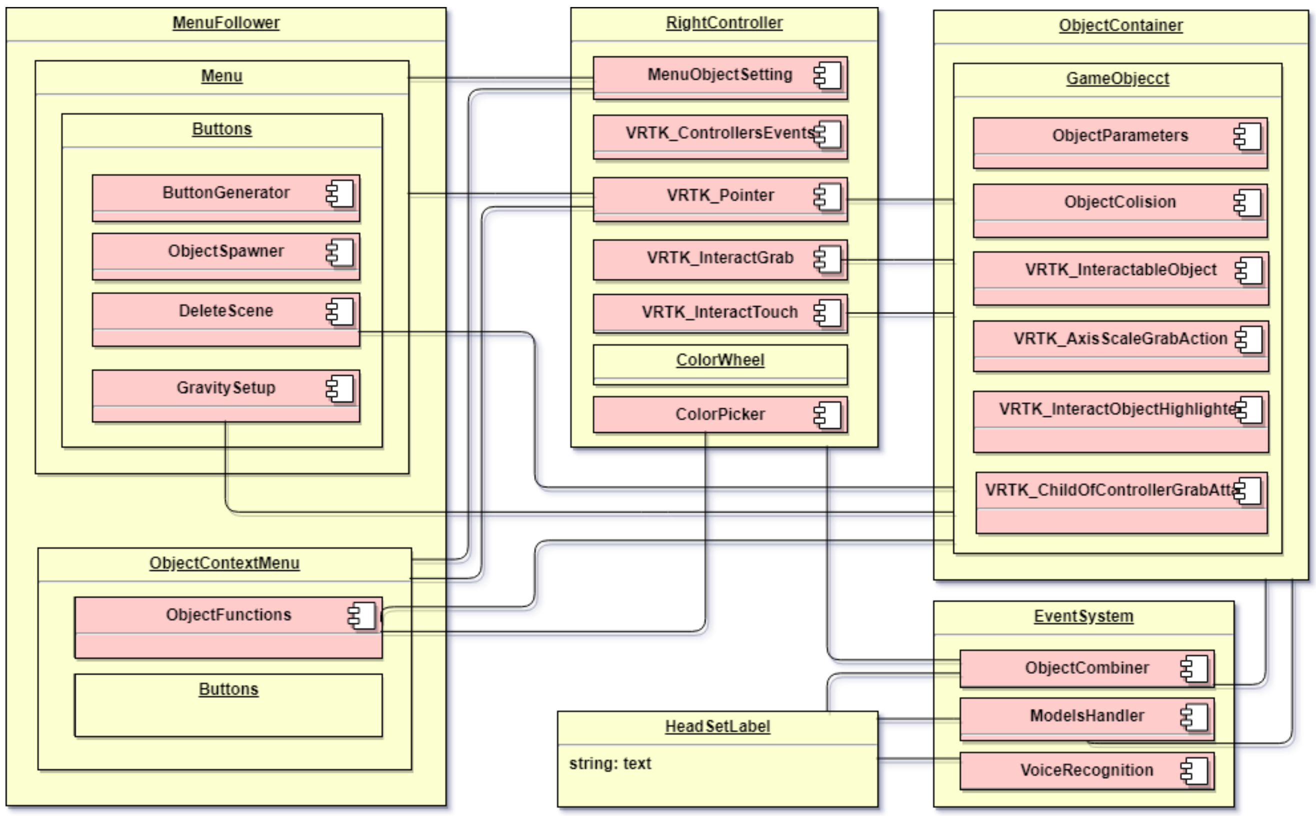




1	Menu button
2	Trackpad
3	System button
4	Status light
5	USB charging adapter
6	Tracking sensor
7	Trigger
8	Grip button

Img 2: HTC controllers (http://notes.caseorganic.com/2016/11/02/htc-vive-controllers/)

Implementation



Img 3. VR module model

Dynamics

The VRTK_ControllersEvents component detects all actions of the HTC Vive controllers buttons.

- **Turning the VR laser pointer**

While the button 2 on the controller is pressed, VRTK_PointerScript calculates the collision of the ray from the controller with the other objects in the scene. This ray is rendered by another component.

- **Showing the menu**

When button 1 on the HTC Vive controller is pressed, the MenuObjectSettings component shows the main Menu with the buttons in the scene. To hide the menu, button 1 needs to be pressed again. The available network architecture names and all object categories of the actual network are refreshed and stored in lists in the ModelsHandler component. The ButtonGenerator component will generate buttons with the object category name and actual architecture.

- **Spawning the GameObject**

The category of the object can be selected by the VR laser pointer in the main Menu and confirmed by the trigger button (button 7). The selected object is generated by Importer module and spawned in the scene by the ObjectSpawner component. Object category and noise are also stored in ObjectParameters component.

- **Changing the network architecture**

Network architecture can be changed in the Menu by choosing the button with the actual architecture name. Available architecture names are stored in list ModelsHandler. ButtonGenerator component calls the ModelsHandler method ChangeArchitecture() to return next architecture name and list of its object categories. The object category buttons are deleted and regenerated and the architecture button name is updated.

- **Scaling, rotation and moving with objects**

The user can grab the object with the controllers to change its position, scale, and rotation. To grab the GameObject the user needs to hold button 8. The GameObject is then assigned to the controller by the VRTK_ChildOfControllerGrabAttach component and follows its position and rotation. Scaling action is provided by the VRTK_AxisScaleGrabAction. The user needs to grab the GameObject with both of the controllers and move them further/closer to increase/decrease the scale. The collision between the controller and the GameObject is calculated by the VRTK_InteractTouch.

- **Display the object context menu**

The user can select the object by pointing the VR pointer on it and pressing the trigger button. The selected object is highlighted to red color by VRTK_InteractObjectHighlighter. The main Menu is turned off and it is replaced by ObjectContextMenu. The selected object is assigned to ObjectFunctions component.

- **Changing object gravity**

The gravity of the selected object can be turned on/off by the GravityButton in the ObjectContextMenu, which calls the method TurnOnGravity() in the ObjectFunctions component.
The gravity of all generated objects in the scene can be turned on/off at once by GravityButtons in the main Menu by GravitySetup component.

- **Changing object color**

Color of the selected object can be turned on/off by the ColorButton in the ObjectContextMenu, which calls the method ColorPicker() in the ObjectFunctions component. ColorPicker() shows the ColorWheel palette on the touchpad of the HTC Vive controller. The user can choose the desired color on the touchpad.

- **Deleting the GameObject**

Selected Object can be deleted by the DeleteButton in the ObjectContextMenu, which calls the method DeleteGameObject() in the ObjectFunctions component.
All generated objects can be deleted at once by DeleteButton in the main Menu by DeleteScene component.

- **Combining objects**

The user can combine 2 objects of the same category while holding them with the controllers and connecting them. The parameters that the objects are held by controllers are stored in ObjectParameters component by ObjectCombiner. The ObjectColision component calculates the collision of 2 objects held by controllers. When the collision between these 2 objects is detected the CombineObjects() method in the ModelsHandler component is called and the network generates new object. The colliding objects are deleted and new object similar to 2 previous is spawned.

- **Headset feedback label**

The HeadSetLabel shows the user a message when the object is being generated. The message is edited in ModelsHandler and ObjectCombiner component. HeadSedLabel also shows the recognized text spoken by the user, which is edited by VoiceRecognizer component.

- **Voice control**

Application also supports voice control. To start recognition user needs to say Charlie, which is name of our custom service. When recognition is in progress, user is provided with visual feedback. Supported commands includes generating new objects and changing architecture of neural architecture currently in use.

2.2 Importer

Analysis

This module is responsible for importing new 3D models which are generated at runtime by the end user. into Unity game engine. These 3D objects are also placed into the scene as GameObject so the user can interact with them. Being able to create and import new 3D models at runtime is essential for our project. We want to allow the user to create new objects while he's inside a VR environment. We could just bundle the final product with 3D GAN neural network, pretrained models and all other modules, but running all these modules like VR application, object generation and object processing all at once at one machine would put the client machine under huge load and consume a lot of hardware resources.
Therefore, we've decided that the best solution will be to create the object on a remote dedicated server (or servers) and provide a service, which can be called by the client machine when needed. This way the client machine doesn't have to spend resources on generating the 3D object, but a reliable and fast object importer is needed for downloading and importing the newly generated object into the VR environment.

Design

The client sends a request using our predefined REST API requesting an object from a specific category. After receiving a successful reply, the reply contains the 3D object data in a wavefront object file format, which is then processed by the object importer and transformed into a GameObject instance. The new GameObject if afterwards placed in the scene, where the user can interact with the objects.

Most of the hard work, that is generating the object using a neural network and converting it into a usable format, is being done outside of the Unity game engine and the end user equipment.

With this design we aim to reduce the load on the end user equipment and achieve a smoother and overall better experience for the end user.

Implementation

The client sends a HTTP POST request to the 3DSpaceGen Flask Server containing a JSON object with the specification for the requested object. The Flask Server forwards the request to the Accenture Inference machine, which generates a 3D object using a pre-trained model based on the parameters included with the request. After the object generation phase, post-processing follows. The generated object is converted from the output .MAT file to wavefront object file (.OBJ), and a second pass of post-processing minimizes the face and vertex count. The object is then sent back to the 3DSpaceGen Flask Server, which forwards the object to the client machine, running the VR setup. The client machine imports the object file and adds it into the scene.

2.3 Healthcheck

We used Healthchecks.io for lightweight server monitoring: ensuring a particular system service, or the server as a whole is alive and healthy.

We created a list of checks one for each client and the server. Each check has a unique "ping" URL. Whenever we make an HTTP GET request to this URL, Healthchecks.io records the request and updates the "Last Ping" value of the corresponding check.

When a certain, configurable amount of time passes since last received ping, the check is considered "late". Healthchecks.io then waits for additional time (configured with the "Grace Time" parameter) and, if still no ping, sends us an alert.

As long as the monitored service sends pings on time, we receive no alerts. As soon as it fails to check in on time, we get notified by slack, that we used as integration.

2.4 Rest API Service

Design

We designed a Node.js server as a REST service that is responsible for responding to a user request, where a user specifies a kind of operation he wants to perform. In order to identify requests, we designed the API using Swagger.

Since our server is not able to generate an object (missing graphics card), we propose to use a WebSocket server in order to communicate with the external machine, that has appropriate hardware to perform this task. Every client connected to the server has to be authenticated and has to send information about model architecture, objects, and operations related to them.

When the request from the client comes, the server determines which type of operation is performed with a given object. Every request contains various information, such as requested object type (e.g. chair, table...), operation or message identifiers. This information is further transformed into JSON format and sent to the inference client according to request parameters.

The server is then waiting for the response from the inference client. If the inference server returned the generated object as the response, the server sends this object to the end user, otherwise sends an HTTP error message.

Implementation

The server is written in Node.js using the Express framework and is responsible for handling user requests. We defined routing using methods of the Express app object that correspond to HTTP methods: app.get() to handle GET requests and app.post() to handle POST requests. To view all routes, see *Attachment C - API documentation*.

When the server is running, it waits for a remote client connection. WS client will try to establish a connection with the server. After the connection is established, the client sends a handshake message that includes authorization data and information about services and models, it provides. If the client isn't authorized, the connection is closed by the server. Otherwise the server stores information about a client.

Model pre-processing

Analysis

The 3D model is generated by our implementation of 3D GAN neural network, which outputs a .MAT file format. This file format is not supported by Unity. Another problem with the generated objects is that it contains a huge number of vertices, with many of them being redundant. Therefore, we have identified 2 main problems, which need to be addressed before forwarding the generated model to the client machine. These problems are:

- Incompatible object file format
- Too many redundant vertices

Design

After the 3D object is generated, this model pre-processing module is called. This module consists of 2 parts – an object format converted and an object vertex count minimizer.
First, the object format converter is initialized. This algorithm builds a wavefront object file in standard .OBJ format from the source .MAT file. The resulting object file consists of vertex coordinates and face data – a vector of three indexes indicating which vertices make up the face.
Afterwards, the object simplification algorithm is called, which iterates over the vertices of the object and removes redundant vertices. It is also essential to reconstruct the faces after the vertices have been removed, which is taken care of by this algorithm as well.
The result is a wavefront object file, which is compatible with Unity game engine, with optimized vertex count.

2.5 Inference clients

We employ three different inference clients in our application and each provides different functionality. Behind each of the clients is a different architecture of neural network.

- **3D-GAN -** Provides the generation and combination of existing objects.
- **3D-IWGAN –** Provides only generation of objects. These objects tend to be of better quality than 3DGAN.

- **VAE-3D-GAN –** Provides generation of an object. This object can be conditioned with description, e.g. ‘garden chair’.

Each of the clients is dockerized and can be easily run anywhere. The manual for startup is provided in ***README.md*** file.

Repositories in TFS*:*

{3D-GAN, 3D-IWGAN, VAE-3D-GAN}
- *Contains training scripts*

{3D-GAN, 3D-IWGAN, VAE-3D-GAN}-Inference-Client
- *Contains code for the inference websocket clients in python*

{3D-GAN, 3D-IWGAN, VAE-3D-GAN}-Trained-Models
- *Contains trained models for furniture*

Basic repositories contain the training scripts for models. 3D-GAN network is implemented in PyTorch, while the 3D-IWGAN and VAE-3D-GAN are implemented in Tensorflow. The *-Trained-Models repositories only contain the trained models.

The architecture of the generative adversarial neural network is implemented as specified in paper by Wu et al. From NIPS 2016 conference. The architecture and training scripts are modified version of open source repositories.

2.5.1 Client module

Each of the inference client modules is a simple web socket client implemented in python. On the startup, this WS client will try to establish a connection with our specified school server (IP address and port are provided as command line arguments). After the connection is established, the ‘client’ will listen to requests made from ‘server’ in an endless loop. In the handshake message, the client informs the server about its services and models, it provides.

The request message from server is a JSON object. The client requires different fields for each service.
- On the request message with service_type - *‘generate’*, inference module is called for inference of random object. Type of object is required in the field params.
- On the request message with service_type - *‘combine’,* inference module combines the noises given in *‘noise’* fields, and then does inference on combined noise. Type of object is required in the field params.
- On the request message with service_type - *‘reconstruct’*. The client requires the object type and URL of an image. The client downloads the URL from web and generates on it.

The generated object is a 3D voxelised object with size 32x32x32. Afterwards it is encoded into a standard *obj* format, which contains the vertices, faces etc. This format is returned in one long string.

In case of an exception thrown or any error, the client returns the response with only error field with error message and id of a request.

2.5.2 Preprocess module

To increase the quality of objects generated we have employed the two preprocess methods.

Remove artifacts removes the noise around the generated object. The noise is in form of small independent clusters of cubes around the main object. We employ the graph processing algorithm to indentify them and find their size. Then we remove all the components which are smaller than 1% of the largest component.

Connect components method to connect the independent components. Each component connects to the main (biggest) component. At first, we find the two closest points between components, and then we connect them with brasenham algorithm. For example, this solves the issue when the generated chair is not connected with its legs.

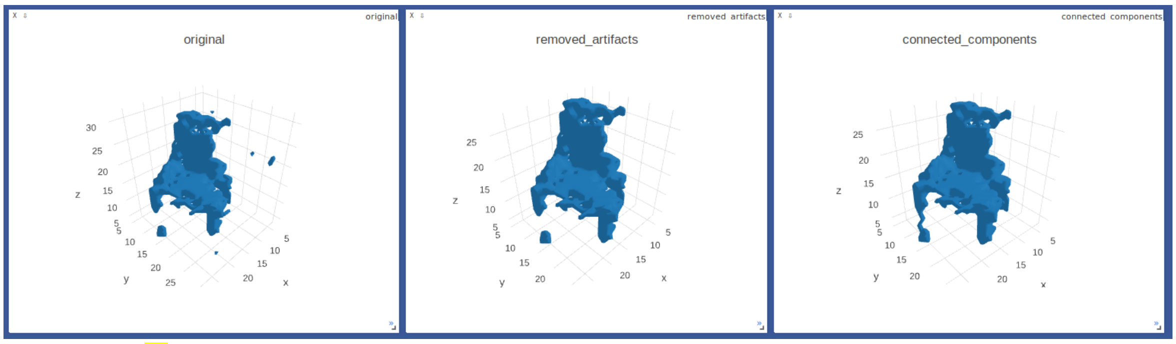


Image n. X. The example of preprocessing method on a chair. In the first section is pure generated chair. The second section contains the removed artifacts. In the third section the front right leg is connected to the main body.

Reference paper:
WU, Jiajun, et al. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In: *Advances in Neural Information Processing Systems*. 2016. p. 82-90.

Reference repositories:
- https://github.com/meetshah1995/tf-3dgan
- https://github.com/rimchang/3DGAN-Pytorch

2.5.3 Model training

Repository: {3D-GAN, 3D-IWGAN, VAE-3D-GAN}

For the model training of 3D-GAN we have implemented a pipeline for training, result visualization and model storage. It uses the **sacred** framework to isolate a single training run. Main script loads a configuration file of training hyperparameters and then starts the training. Then it loads it with the provided dataset. For training, we have used 3DShapeNets dataset of objects, which can be freely downloaded from internet. The dataset consists of thousands of simple 3D objects (e.g. chairs, tables, toilets) in various angles.
When the training is finished, the trained pytorch model is saved into local directory and an agent sends a slack notification to team slack channel about training finish. Also couple of objects are generated and posted to our hosted the **Visdom** server for visualization. The **Visdom** server is a simple docker container running on our school virtual machine. The Visdom is a framework developed by Facebook in python for visualization of various graphs and models.
For the 3D-IWGAN and VAE-3D-GAN we have used a simple non-automated approach with scripts.

Attachment A - Unity manuals

How to set up unity development environment

Set up Unity only for development (without HTC Vive):

1. Download and install Unity version **2018.3.13f1**
2. Download and install .NET Framework version: 4.7.1
3. Clone **3DSpaceGen_Unity** repository from TFS server
4. Open Unity and then open cloned folder
5. Click play to test the program

Set up Unity for development with HTC Vive:

1. Download Steam from official website
2. Download SteamVR version 1.3.23
3. Download and install Unity version **2018.3.13f1**
4. Download and install .NET Framework version: 4.7.1
5. Clone **3DSpaceGen_Unity** repository from TFS server
6. Open Unity and then open cloned folder
7. Click play to test the program

Important: Voice recognition is supported only with Windows 10!

Configuration of the application

The application is configured using configuration file named projectConfig.cfg, located in the root directory in the project. The configuration can be changed in this file, if needed. There are several configuration sections:

- General section contains url to services, our server provides, and the default GAN architecture, which is set up on startup.
- ImageSearch section contains uri to Bing image search api and the subscription key.
- VoiceRecognition section contains url to Luis api, used for the intent recognition from the speech, and the corresponding subscription key for it. There is also Postman token, which is deprecated.

How to build and test Unity project

Requirements

- Windows 10 x64 operating system
- Windows PowerShell
- Unity Editor version 2018.3.13f1 (https://unity3d.com/get-unity/download/archive)
- SteamVR version 1.2.3 (https://github.com/ValveSoftware/steamvr_unity_plugin/releases/download/1.2.3/SteamVR.Plugin.unitypackage)
- .NET Framework version 4.7.1 (https://dotnet.microsoft.com/download/visual-studio-sdks)

- VRTK 3.3.0 (https://github.com/ExtendRealityLtd/VRTK/releases)

Note: VRTK and SteamVR are already part of the project.

Steps

1. Clone 3DSpaceGen_Unity git repository. *git clone https://tfs.fiit.stuba.sk:8443/tfs/StudentsProjects/3DSpaceGen/_git/3DSpaceGen_Unity*
2. (Optional) If you're trying to build the project for the first time, you need to open the scene in Unity Editor first in order for Unity to import all necessary assets.
   - Scene that needs to be opened is located at **3DSpaceGen_Unity\Assets\Scenes\3DspaceGenScene2.unity**
3. Download build script from repository scripts-and-configurations. Script name is **build.ps1** and is located under **scripts-and-configurations\scripts\build.ps1**
4. Run the build script in Windows Powershell. For general help use *Get-Help <path-to-script>* or *Get-Help <path-to-script> -Detailed*
   - (Optional) Script execution policy might need to be changed. Use *Set-ExecutionPolicy Unrestricted* as a temporary workaround in case Windows doesn't allow you to run the script.
   - If you have Unity installed under other than default path (C:\Program Files\Unity\Editor\Unity.exe), you need to specify path to the Unity executable as a string via parameter - *UnityPath <string>*
     You can skip this if you have path to Unity in PATH variable.
   - Path to the Unity project directory needs to be specified via parameter - *ProjectDir <string>*
   - If you wish to execute only tests, please add -*TestsOnly*
   - If you wish to execute only build, please add -*BuildOnly*
   - If you don't want the script to ask for confirmation, add –*Force*

Attachment B - Virtual machine setup

This is a guide for setting up the virtual machine with the 3 clients and a server.

Install docker

At first install docker and docker-compose for ubuntu. Follow the guide in the URL below.
https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-04
sudo apt install docker-compose

CI/CD (Optional)

To setup the agent for TFS CI/CD.

Download
mkdir agentcd agentwget https://vstsagentpackage.azureedge.net/agent/2.136.1/vsts-agent-linux-x64-2.136.1.tar.gztar zxvf vsts-agent-linux-x64-2.136.1.tar.gz

Configure (get key from tfs)
./config.sh

Run as daemon
sudo ./svc.sh installsudo ./svc.sh start

You can trigger the releases in TFS after this.

Setup the server

Clone the repository with scripts and configs.
git clone https://tfs.fiit.stuba.sk:8443/tfs/StudentsProjects/3DSpaceGen/_git/scripts-and-configurationssudo apt-get install apache2

Apache

cd etc/apache2/sites-available/sudo cp ~/scripts-and-configurations/configurations/* .
Rewrite the ip address in http.conf on line XX 51.144.227.194
sudo a2dissite 000-default.conf sudo a2ensite http.conf sudo a2ensite https.conf sudo systemctl reload apache2

Setup env file for server

cat > .env << EOFPORT=8100SERVER_URL=https://51.144.227.194/apiNODE_ENV=developmentEOF

Fix apache deployment

Create self-signed certificate (or register domain, whatever, we need ssl certificate that's the point)

#This creates self-signed x509 ssl certificate
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout mysitename.key -out mysitename.crt

```
#Add certificate to active certificate directory
sudo mv mysitename.crt certifikate.crt
sudo mv mysitename.key private.key
sudo cp certifikate.crt /etc/ssl/certs/certificate.crt
sudo cp private.key /etc/ssl/private/private.key
#If not self-signed, ignore this step. In https.conf comment line SSLCACertificateFile /etc/apache2/ssl.crt/ca-bundle.crt

#Enable proxy, ssl and ws tunneling
sudo a2enmod ssl
sudo a2enmod proxy_http
sudo a2enmod proxy_wstunnel
#restart apache service to apply changes
sudo systemctl restart apache2
```

Clients and server setup

After all is prepared, you can finally start the clients and servers. Continue by reading the README.md files in given repositories.

Attachment C - API documentation

3DSpaceGen API 1.0.0

`[ Base URL: /api ]`

API provides resources which are generated object using pretrained models

Schemes

HTTPS

3DGenerator

GET `/services` Get list of available services for object generation

Parameters Try it out

No parameters

Responses **Response content type** application/json

Code	Description
200	*success*

Example Value Model

```
{
  "architecture": [
    {
      "model": "string",
      "url": "string",
      "methods": [
        "string"
      ]
    }
  ]
}
```

Code	Description
500	*error* **Example Value** Model `{ "code": 0, "message": "string" }`

```
{
  "code": 0,
  "message": "string"
}
```

GET `/{architecture}/{model}` Get generated object by neural network

Parameters

Try it out

Name	Description
architecture * required `string` *(path)*	Name of the architecture used to generate the model
model * required `string` *(path)*	Type of the model to be generated

Responses

Response content type application/json

Code	Description
200	*success* **Example Value** Model

```
{
  "data": "string",
  "noise": "string"
}
```

Code	Description
500	*error* **Example Value** Model `{ "code": 0, "message": "string" }`
503	*Service Unavailable* **Example Value** Model `{ "code": 0, "message": "string" }`

POST `/{architecture}/{model}/combine` Combine two objects to generate their combination

Parameters

Try it out

Name	Description
architecture * required `string` ***(path)***	Name of the architecture used to combine two models
model * required `string` ***(path)***	Type of the model to be generated
RequestBody ***(body)***	**Example Value** Model `{ "noise": [ "string" ] }` **Parameter content type** application/json

```
{
  "noise": [
    "string"
  ]
}
```

Responses Response content type application/json

Code	Description
200	*success* **Example Value** Model `{ "data": "string", "noise": "string" }`
500	*error* **Example Value** Model `{ "code": 0, "message": "string" }`
503	*Service Unavailable* **Example Value** Model `{ "code": 0, "message": "string" }`

POST `/{architecture}/{model}/reconstruct` Generate an object from an image.

Parameters Try it out

Name	Description
architecture * required `string` *(path)*	Name of the architecture used to combine two models
model * required `string` *(path)*	Type of the model to be generated

Name	Description
RequestBody *(body)*	**Example Value** Model

```
{
  "url": "string"
}
```

Parameter content type

application/json

Responses **Response content type** application/json

Code	Description
200	*success*

Example Value Model

```
{
  "data": "string",
  "noise": "string"
}
```

Code	Description
500	*error*

Example Value Model

```
{
  "code": 0,
  "message": "string"
}
```

Code	Description
503	*Service Unavailable*

Example Value Model

```
{
  "code": 0,
  "message": "string"
}
```

Models

```
ServiceInfo    {
   architecture
                           [
                      List of all available models for given architecture
                           {
                          model                 string
                                                Model that will be generated
                          url                   string
                                                URL to the resource
                          methods
                                                    [
                                                List of all available methods
                                                string]
                         }]
}
```

```
ObjectNoise    {
   noise
                           [
                      List of noises that will be combined
                      string
                      noise
                      ]
}
```

```
ImageUrl    {
   url                string
                      URL of an image that generator takes as input
}
```

```
GeneratedObject    {
   data               string
                      generated object in .obj format
   noise              string
                      original noise from wcich object was generated
}
```

```
ServerError    {
   code               number
                      Error code
   message            string
                      Error message
}
```