Faculty of Science and Technology
Department of Computer Science

Freia: Exploring Biological Pathways Using Unity3D

—

Kenneth Knudsen

INF-3990 Master's thesis in Computer Science, November 2015

Abstract

To understand the biological processes related to the development of cancer there is a need for interactive data visualization tools that integrates experimental data with biological knowledge. Existing visualization tools have shown their usefulness, but next-generation biological data analysis requires both integrating different data types and larger datasets. This requires data exploration tools with a flexible data model that is efficiently represented and processed. Entity Component System in game engines fulfil both requirements.

There are two main challenges for using game engines for biological data visualization. First, how to represent the biological data to be visualized using the game engine data structures. Second, how to efficiently implement the necessary data exploration operations using these data structures.

This thesis presents the, to our knowledge, first approach for mapping biological data to the ECS model. We used the approach to implement Freia, an application for visualizing gene expression data integrated with pathway images. We evaluated the performance and scalability of Freia by measuring the smoothness of key data exploration operations. Our results show that Freia provides a frame rate above 30 FPS for these operations for up to 100 simultaneously shown pathways.

We believe our approach demonstrates that game engines are well suited to implement data visualization tools for the upcoming biological data studies.

Acknowledgements

First I would like to thank my advisors, Associate Professor Lars Ailo Bongo, PhD Candidate Bjørn Fjukstad and Associate Professor John Markus Bjørndalen for their continuous feedback, support and motivation during the course of this project.

I would thank the NOWAC research group for their input to this project.

To my fellow students, particularly those in "Slytherin" thanks for all the great years at the university. I will really miss our daily burn runs!

Finally, I would thank my beloved family for encouragement, support and all the warm dinners throughout the period of this project.

Kenneth
Tromsø, November 2015

Contents

Abstract i

Acknowledgements iii

List of Figures vii

List of Tables ix

List of Abbreviations xi

1 Introduction 1
- 1.1 Challenges & Requirements . . . 2
- 1.2 Proposed solution . . . 3
- 1.3 Contributions . . . 3
- 1.4 Outline . . . 4

2 Mapping 5
- 2.1 Entity Component System (ECS) . . . 5
- 2.2 Biological Pathways and data . . . 7
- 2.3 Mapping from biological data to ECS . . . 9
- 2.4 Lesson Learned . . . 10

3 Freia 13
- 3.1 Visualization of Biological Pathways . . . 14
- 3.2 Operations . . . 17
  - 3.2.1 Pathway Search . . . 17
  - 3.2.2 Gene Search . . . 19
  - 3.2.3 Gene Expression Visualization . . . 19
  - 3.2.4 Path Discovery . . . 20

4 Evaluation and discussion 23
- 4.1 Experimental Setup . . . 24
- 4.2 Unity3D Profiler . . . 24
- 4.3 Experiments . . . 27

4.3.1 Load Pathways 27
4.3.2 Interaction with Pathways 28
4.3.3 Using a Game Engine for Visualization of Biological Data 31
4.4 From 2D to 3D Visualization 33
4.4.1 Pathway connectivity in KEGG 33

5 Related Work 35
5.1 Kvik 35
5.2 KEGG 36
5.3 BioCarta 37
5.4 KEGGViewer 38
5.5 Caleydo 39
5.6 Entourage 40
5.7 Dynamic Exploration 41
5.8 UnityMol 42

6 Conclusion 43
6.1 Future Work 44

Bibliography 47

List of Figures

2.1 An entity 5
2.2 Drawing system 5
2.3 ECS overview of Super Mario. 6
2.4 Component alignment in memory. 7
2.5 KEGG pathway for estrogen signaling. 8
2.6 Microarray technology. 9
2.7 ECS table overview a pathway visualization application. . . . 10

3.1 Screenshot of Freia. 13
3.2 Visualizing gene expression data on KEGG pathway maps . . 15
3.3 Architecture of Freia. 16
3.4 Pathway search results from the search term «Cancer». . . . 17
3.5 Pathway search results to the search term «Cancer». 18
3.6 Visualizing gene expression data on KEGG pathway. 20
3.7 Path discovery in Freia. 21

4.1 Effects of frame rate on user perception. 25
4.2 Builtin profiler in Unity3D. 26
4.3 Screenshot of Unity3D. 30
4.4 Pathway connection clustering. 34

5.1 Screenshot of the renin-angiotensin pathway (KEGG pathway id hsa04614) in Kvik Pathways. 36
5.2 KEGG pathway for estrogen signaling. 37
5.3 Screenshot of the Wnt signaling pathway in BioCarta. 38
5.4 Overview of the KEGGViewer component 39
5.5 Caleydo visual analysis framework for gene expression data in its biological context. 40
5.6 Entourage showing the Glioma pathway in detail and contextual information of multiple related pathways. 40
5.7 Dynamic exploration of two pathways with connected edges. 41
5.8 Comparing a cytoscape visualization to a UnityMol visualization. 42

1 Screenshot of profiler with one pathways with drag operation. 52
2 Screenshot of profiler with one pathways with gene expression operation. 53
3 Screenshot of profiler with 7 pathways with drag operation. . 54
4 Screenshot of profiler with 7 pathways with gene expression operation. 55
5 Screenshot of profiler with 14 pathways with drag operation. 56
6 Screenshot of profiler with 50 pathways with drag operation. 57
7 Screenshot of profiler with 100 pathways running. 58

List of Tables

2.1 Gene expression result from microarray technology. 9

3.1 REST Interface to the KEGG database of Kvik framework[7]. 16

4.1 Pathways used to evaluate Freia. 27
4.2 Time to load pathway visualization. 27
4.3 Time to load pathway visualization from Kvik Pathways[7]. . 28

List of Abbreviations

API	Application Programming Interface
CPU	Central Processing Unit
DNA	Deoxyribonucleic acid
ECS	Entity Component System
FPS	Frames Per Second
FTP	File Transfer Protocol
GIF	Graphics Interchange Format
KEGG	Kyoto Encyclopedia of Genes and Genomes
KGML	Kegg Markup Language
LOC	Lines of Code
mRNA	Messenger ribonucleic acid
NOWAC	Norwegian Women and Cancer
PNG	Portable Network Graphics
REST	Representational State Transfer
XML	eXtensible Markup Language

1 Introduction

Cancer is a complex group of diseases with many possible causes. It is the second leading cause of death in the United States, and it is estimated to pass heart disease death rate in a few years. The increase in cancer cases is due to the world population ages and increases in size. The probability of being diagnosed with cancer for men and women is 43% and 38%, whereas it is only 3.4% and 5.4% for men and women younger than 50 years [1]. The authors of [1] estimate that about 589,430 Americans will die from cancer this year, which corresponds to about 1,600 deaths per day.

The NOWAC epidemiological study was founded to understand development of cancer, effect of exposes, and to find clinical diagnosis and treatments [2]. NOWAC has acquired questionnaire information from 170,000 women and a bio bank with more than 60,000 blood samples. Analyzing the blood samples and questionnaire information may expose commonalities with patients, does smoking or specific diets affect the risk of cancer? However, analyzing the amount of data is challenging, due to the complexity of data and the disease. Therefore, there is a need for a visualization tool which can help the researchers expose these commonalities.

The main focus of the thesis is exploring how well suited game development frameworks are for such biological data visualization.

1.1 Challenges & Requirements

In data-driven studies such as NOWAC, there is a need for an agnostic approach to explore biological processes to find patterns that help finding and understanding hypotheses about the development of cancer. Such an approach should have several different ways to explore the data and integrate it with known biology. One particular useful approach is using pathways that represents biological processes. Researchers use pathways to investigate patterns in gene expression data in the context of know biological processes. Viewing one pathway at a time may not expose correlation with other pathways, therefore it is vital that the tools supports multiple pathway visualizations.

In collaboration with researchers in epidemiology and computer science we have identified several challenges and requirements for biological data exploration tools for studies such as NOWAC. The tools should be fast and responsive to user inputs and it should use the visual representation such as KEGG[1] or Reactome[2].

The tool should support whatever hardware and software the researchers use. The tool should adapt to the screen resolution the researchers have available, this gives the notion of the same user experience whether or not the researcher has the newest high-resolution monitor.

The tool should be portable to whatever device the researchers have available, and this may include smartphones. Therefore, the tool has to be lightweight and requires a backend for computation and storage.

The life science community often use R[3], Microsoft Excel[4] or BioJS[3] for visualization of biological data. There are projects that specializes in visualization of biological data, but these are tailored for special tasks or data sets which results in few users. This is elaborated in Chapter 5. On the other side, the gaming community has to continuously create and update software which supports and exploits the latest hardware to stay competitive in the gaming industry. The life sciences do not have the user base and commercial financial needed for such frameworks.

We believe a game engine or visualization frameworks can satisfy the requirements above. We therefore investigate how these frameworks can be used for life science data exploration and visualization.

1. kegg.jp
2. reactome.org
3. r-project.org
4. products.office.com/nb-no/excel

1.2 Proposed solution

Our main hypothesis is therefore that game engine developer environment and ecosystems have advantages over the traditional environment used for biological data visualization.

There are several popular game development frameworks such as Unity3D[5], Unreal Engine[6], CryEngine[7], Blender Game Engine[8], Corona SDK[9]. These are all free for non-commercial use. We chose to use the Unity3D game engine because of its ability to deploy a single application for multiple platforms with minimal programming effort. It is used for implementation of both large-scale big budget games to low budget mobile games. The developer community is very active and the developer forums are full of helpful threads. Unity3D also has an asset store where there are many assets such as game avatars or textures, and plugins which can aid the development of the application.

The Unity3D game engine is also based on component system which follows the composition over inheritance. The advantage of ECS is that it creates more flexibility in every entity created because every entity can have the same component, such as a collision script, whether it is a tree, a bullet or a player it may use this script. An entity can be viewed as a key with an unique entity ID, and the entity has components assigned to it. In order for an entity to be calculated on by a system, the entity must contain the required components to the given system and effective execution. The ECS concept is further described in Chapter 2.1.

1.3 Contributions

We implemented a data exploration application, called Freia, using Unity3D. The computation therefore has to be done in the backend. We use the Kvik framework [4] to handle data and computation. This results in a lightweight application that the researcher can use, since the Kvik framework provides the required functionality for retrieving pathway maps and its respective meta-data. The Kvik framework also contains the functionality for accessing experimental data, such as gene expression or methylation.

In order to use the efficiency advantage of the ECS model, the biological data

5. unity3d.com
6. unrealengine.com
7. cryengine.com
8. blender.org
9. coronalabs.com

has to be restructured into entities, components and systems.

The main contributions of this work are:

- To our knowledge, the first approach for mapping the data types of biological data visualization tools for efficient representation, processing and visualization using ECS in game engines.

- A demonstration of the approach by implementing the Freia application for exploring pathways.

- An evaluation of Freia using KEGG and gene expression data.

We found that our approach has several advantage. Particularly when is comes to using a game engine that enforces the use of ECS, because of its modularity and the reusability of the routines and scripts in the game engine. A script or a routine can be how an image should behave in the application, and this can be reused whether it is an image from different image sources. Finally, the greatest advantage by our approach is the efficient processing provided by ECS. ECS aligns structures of the same type together in memory, iterations may exploit spatial locality in memory layout.

However, there are also some disadvantages. Unity3D game engine has a free to use model, but closed source so we do not in detail know the underlying details of for example ECS memory alignment.

We believe our approach demonstrates that game engines are well suited to implement data visualization tools for the upcoming biological data studies.

1.4 Outline

The remainder of the thesis is structured as follows.

Chapter 2 describes the mapping of biological data.

Chapter 3 describes design, implementation and use of Freia.

Chapter 4 evaluates Freia.

Chapter 5 discusses related work.

Chapter 6 concludes and outlines future work.

2 Mapping

In this chapter we describe the ECS model and how it used in game development. Then we describe how to map the biological data to the ECS model, and the challenges and issues doing this.

2.1 Entity Component System (ECS)

An ECS is an architectural pattern which is often used in game development. To our knowledge it has not been used for biological data visualization before. It follows composition over inheritance which creates more flexibility in every entity created, because every entity can have the same component. Entities such as trees, bullets or players may use components as a collider component.

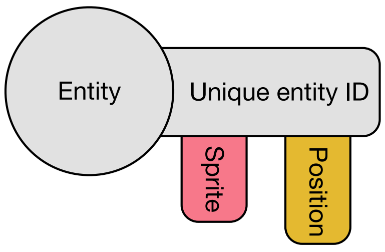


Figure 2.1: An entity

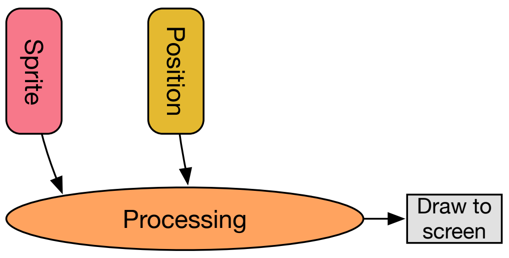


Figure 2.2: Drawing system

An entity can be viewed as a key with a unique entity ID, and the entity has components assigned to it as seen in Figure 2.1. For a system to process on an entity, the entity must have the components that the system requires. As seen in Figure 2.2 the drawing system requires the sprite and the position components in order for an entity to be rendered to the screen. The key may unlock the calculation «key-hole» if it has the right components, «key-pins». An entity may acquire components dynamically, for example in «Super Mario» Mario can acquire the «Super Mushroom» which grows Mario to a bigger version of himself. If Mario is hit by a turtle shell when he is Super Mario, he will go back to normal size. As seen in Figure 2.3 Mario starts with the «Small Mario» component, when Mario hit the Super Mushroom entity he acquires the Super Mario component and lose Small Mario component.

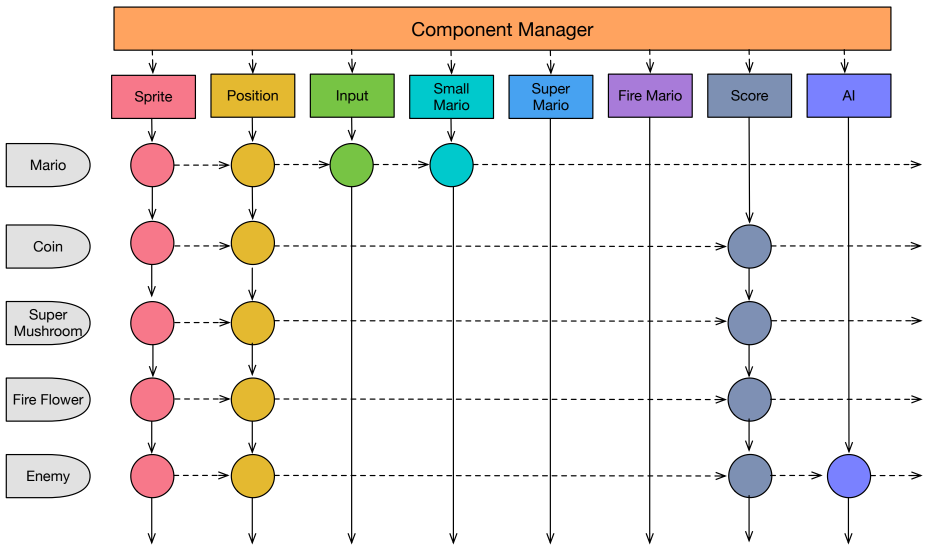


Figure 2.3: ECS overview of Super Mario.

Figure 2.3 shows a small representation of how Super Mario could be represented in a ECS model. Mario is the only entity that has an input and is therefore the only entity that the player can control. Coin, super mushroom, fire flower and enemy entities have all the score component. But the points gained from these entities are not equal. Super mushroom and fire flower is worth of 1000 points and coins are worth 200 points, this is set when attaching a new score component on the entities. It is the system that contains this game logic, the entity only states which components it has, and the components states e.g. a position component stores the x and y position.

The component-based design makes it easy to implement new kinds of entities in contrast to hierarchical design. The authors [5] of argue that a code base is growing in size and complexity and requires frequent refactoring.

Components are loosely coupled, so components are not aware of other components. To reference other entities, all that is needed is the entity ID, there is no need to store a pointer or the entity itself.

When implementing an ECS, components of one type are stored in a large contiguous array. The components of the same type are aligned in memory so the system can sequentially iterate over the array, avoiding jumps in memory and cache misses thereby processing effectivity.

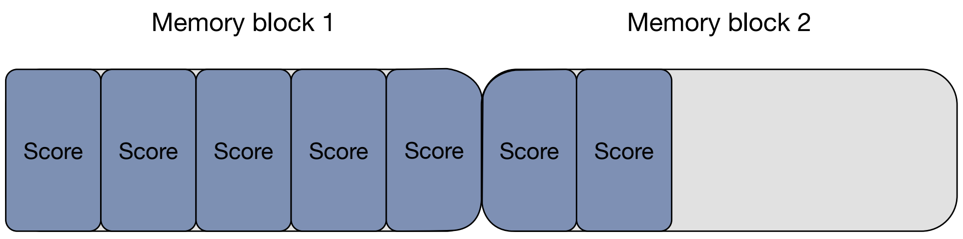


Figure 2.4: Component alignment in memory.

Unity3D does not explicitly follow the ECS model, but it shares similarities with the ECS model. It uses game objects as the unique entity and it attaches components to that entity, e.g. the built-in component «*Transform*». The transform component stores the information about the size, position, scale and scale of the game object.

2.2 Biological Pathways and data

In this project we have focused on KEGG pathways and gene expression data, since these are used to explore NOWAC data. The manually drawn KEGG pathway images are networks of biomolecules and biochemical reactions that describe a series of actions leading to specific biological effects. Figure 2.5 shows an example pathway.

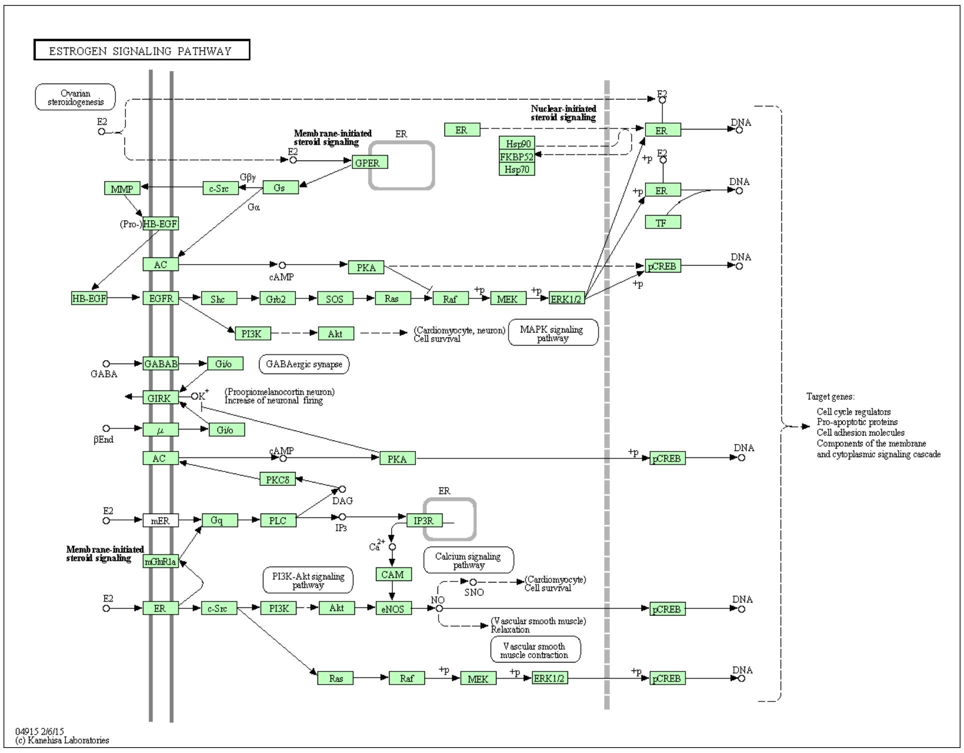

ESTROGEN SIGNALING PATHWAY
Ovarian steroidogenesis
E2
Membrane-initiated steroid signaling
ER
GPER
Nuclear-initiated steroid signaling
Hsp90
FKBP52
Hsp70
DNA
MMP
c-Src
Gs
Gβγ
Gα
(Pro-)HB-EGF
AC
cAMP
PKA
pCREB
TF
HB-EGF
EGFR
Shc
Grb2
SOS
Ras
Raf
MEK
ERK1/2
+p
PI3K
Akt
(Cardiomyocyte, neuron) Cell survival
MAPK signaling pathway
GABA
GABAB
Gi/o
GABAergic synapse
GIRK
K+
(Proopiomelanocortin neuron) Increase of neuronal firing
βEnd
μ
PKCδ
DAG
mER
Gq
PLC
IP3
IP3R
Ca2+
Calcium signaling pathway
mGluR1a
PI3K-Akt signaling pathway
CAM
eNOS
NO
SNO
(Cardiomyocyte) Cell survival
(Vascular smooth muscle) Relaxation
Vascular smooth muscle contraction
Target genes:
Cell cycle regulators
Pro-apoptotic proteins
Cell adhesion molecules
Components of the membrane and cytoplasmic signaling cascade
04915 2/6/15
(c) Kanehisa Laboratories

Figure 2.5: KEGG pathway for estrogen signaling, rest.kegg.jp/get/hsa04915/image.

The small rectangles in Figure 2.5 represents genes and the arrows from or to the genes are links. The rounded squares are other pathways that this pathway is connected to. The small circles are compounds which is a collection of small molecules, biopolymers, and other chemical substances that are relevant to biological systems.

The content within the KEGG pathway images are represented in separate file as Kegg Markup Language (KGML). The KGML file is describing the type of the content, coordinates, size of the content and if there are any links they are listed up. Not all the KEGG pathway image content is represented in the eXtensible Markup Language (XML) file and the information about the content may therefore be lacking vital information, e.g. annotated data such as the cell membrane that is the two vertical lines in Figure 2.5.

The NOWAC biobank is a collection of genetic data from patients and is currently consisting of 70 000 blood samples. The genetic information is encoded in the Deoxyribonucleic acid (DNA) in units (genes) and selectively used (expressed/transcribed to Messenger ribonucleic acid (mRNA)) as templates for production of proteins, which in turn are the working units of the cell [6].

Microarray technology is used to measure all levels of information flow from biological levels. The Figure 2.6 is showing the full process of extracting gene expressions microarrays. The gene expressions microarrays are used for three main purposes: i) identification of differentially expressed genes, ii) class discovery (grouping of samples based on gene expression profiles), and iii) class prediction (assignment of new samples to pre-defined groups) [6].

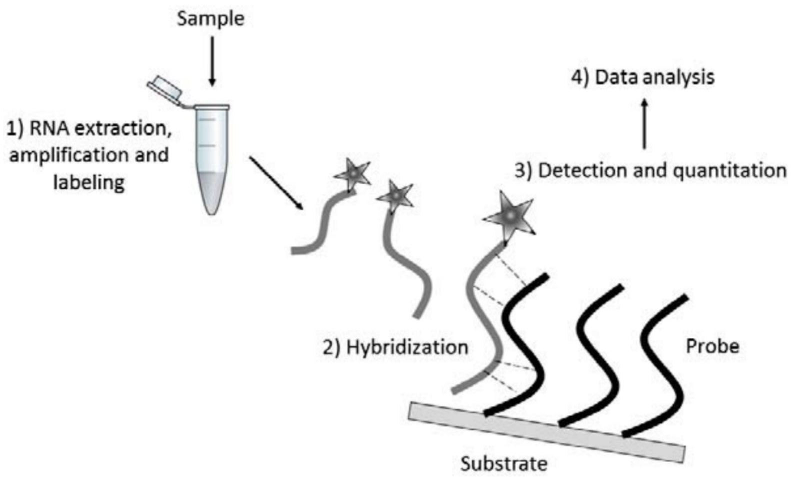

Sample
4) Data analysis
1) RNA extraction, amplification and labeling
3) Detection and quantitation
2) Hybridization
Probe
Substrate

Figure 2.6: Microarray technology. Figure 8 from [6].

The results from microarray technology are represented as seen in Table 2.1.

Table 2.1: Gene expression result from microarray technology.

Id	Expression
PCAT7	1.20
PCAN-R2	-1.34
CASC18	10.62
CASC20	4.00

2.3 Mapping from biological data to ECS

We have mapped the KEGG and gene expression data into seven components and three entities which is shown in Figure 2.7. On top of all the components is the component manager which consist of the systems that interact with the different components. The three entities are to the left and each circle represent which component this entity has. As the arrows in Figure 2.7 points out, adding a new entity or component is easily added at the end of the table.

The component manager contains system such as a drawing system, this system traverses the components and which which of the entities that holds both a sprite and a position component. If we were to add another entity that should be drawn it would create no more involvement in the codebase with the drawing system, since all it has to do is to add the sprite and the position component.

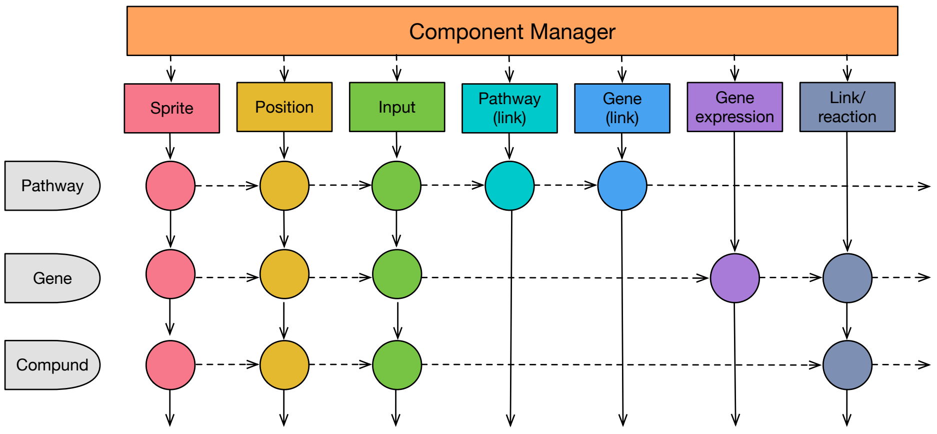


Figure 2.7: ECS overview of a pathway visualization application. The columns are components and each row is an entity.

The ECS methodology is quite new compared to the hierarchical methodology which is what we could have used instead. The ECS model have loosely coupled components, and when the support for several pathway database this would theoretically create less mess.

But as mentioned earlier we have only mapped one of the popular pathway databases in this project. Further down in the development we could see ourselves including other pathway databases which would make us refactoring the codebase whenever we want to scale the information basis.

2.4 Lesson Learned

We experienced some challenges with mapping the biological data to the ECS model. Instead of all the information is inside the one object it is spread across components. In addition, does the pathway, gene and compound entities share component. E.g. all these entities have input, sprite and a position, but only the pathway entity should be able to be repositioned. Genes and compounds are connected to the position within the pathway, so should the position component in the pathway entity push the new information to these position components that are linked to the pathway entity, or should all this be handled in the system? Components should be independent, so therefore the system should handle this.

The ECS model is hard to start with when you quite not fully understand the concept, but again it is even harder to go from the hierarchical model to the ECS model. However, we believe the simplicity of the final implementation makes the mapping worthwhile.

/3 Freia

The mapping in previous chapter is realized as a standalone application, Freia implemented in Unity3D. It uses the Kvik framework [4] as a backend. The application is portable to devices including mobile phones, browser, windows, mac, and linux.

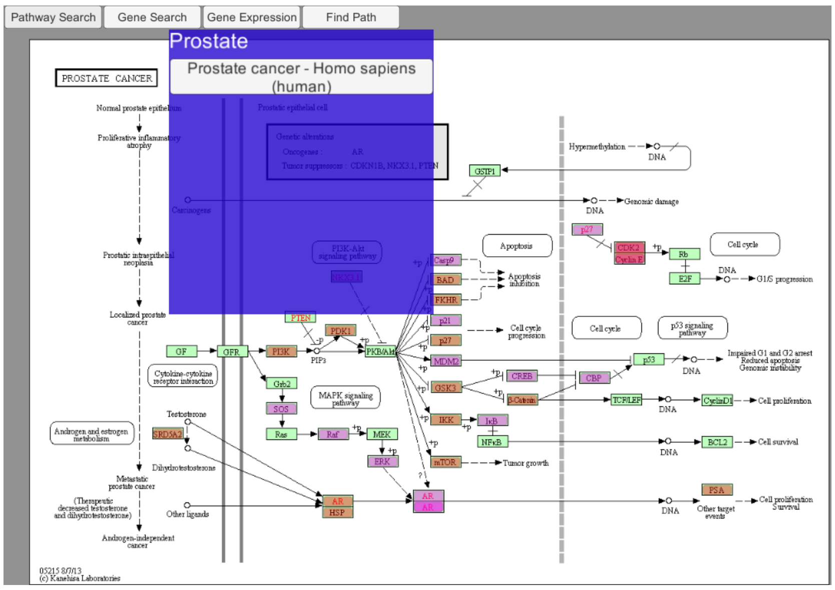


Figure 3.1: Screenshot of Freia. Showing the prostate cancer (hsa05215) pathway with gene expression data.

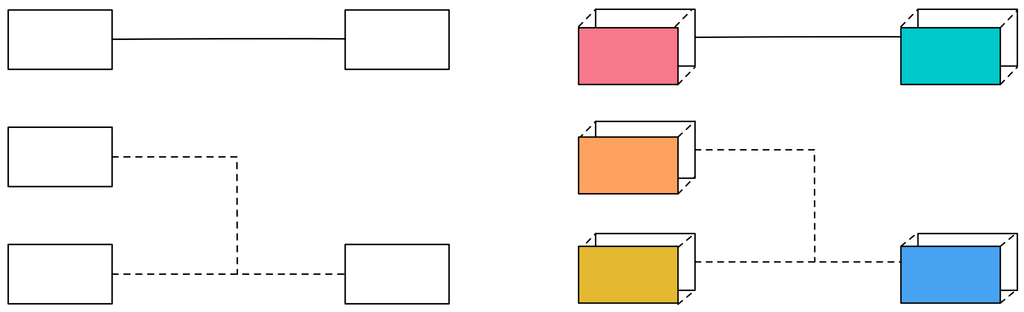
Freia is designed for users to explore human pathways by visualizing Portable Network Graphics (PNG) images. The user may drag these images to any given position and zoom in and out. New pathways may be opened within a pathway if it has a pathway link. Users can highlight genes either by hovering the mouse above the genes, clicking on the genes, or search for genes with the search gene function. Freia view the NOWAC bio bank data by using a gene expression data visualized on top of the pathway images. The genes are colored red or blue depending on whether the gene is up-, or down regulated. Reactions between the genes are marked as links between the genes. Freia supports search within a pathway to determine if there is a link between the genes. Paths between genes are highlighted in blue.

Freia is implemented as a prototype for evaluation of mapping biological data to the ECS model. It is not intended for use by researchers in its current implementation. However, it has the functionality, visualizations and operations commonly used for biological data exploration. We believe Freia can be extended to be useful for exploring real data by changing the simulated data to real data.

The application was designed for the researchers exploring gene expression data in the context of biological pathways. This applies to the operations which are further explained in Chapter 3.2. We demonstrate the functionality of Freia in this YouTube video: `youtu.be/22XmfSYOw08`.

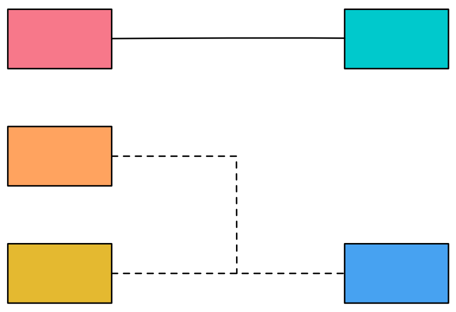
3.1 Visualization of Biological Pathways

The images used in Freia are biological pathways from the KEGG databases are manually drawn images. To make the pathway maps interactive we add a layer on top of these images (Figure 3.2). This layer is built from information about image nodes represented in a XML-like format, KGML. The KGML contains information about contents in the pathway image, such as where the pathways, genes and compounds are located within the image. The KGML representation contain coordinates and size of the nodes.



(a) Original static pathway image from KEGG.

(b) Overlaying graph nodes from the KGML representation of the pathway.



(c) Finished visualization.

Figure 3.2: Visualizing gene expression data on KEGG pathway maps. Figure is inspired from Figure 5.5 from[7].

With the knowledge of where the nodes are in the image, it is possible to create nodes on top of nodes in the image. The user does not see these overlaid nodes as they are transparent and it creates the illusion that the image is interactive.

As Freia was intended to be a light-weight application, it had to have a backend which does processing and storage. Freia uses the Kvik framework [4]. It is tightly coupled with the backend and is therefore dependent on a network connection for the visualization of the biological data. Kvik framework uses the KEGG database for the pathway images and NOWAC bio bank for the gene expression data. The architecture is summarized in Figure 3.3.

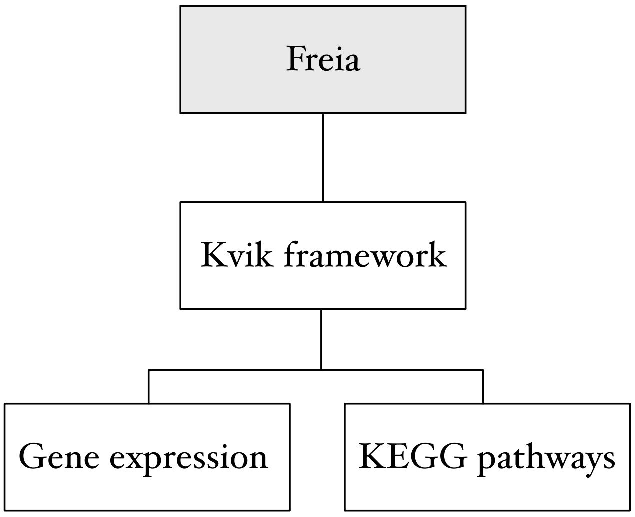


Figure 3.3: Architecture of Freia.

Kvik framework provides a REST interface to access the KEGG database (Table 3.1). In Freia we use simulated gene expression values on a scale from 0 to 1.

Table 3.1: REST Interface to the KEGG database of Kvik framework[7].

Resource	Description
GET */pathway/*{*id*}*/json*	Returns the KGML of the id in json format.
GET */public/pathways/*{*id*}*.png*	Returns the png image of the id to the given pathway.
GET */search/*{*term*}	Returns pathways that matches the given term in json format.

Freia is made in Unity3D with a single scene, the camera is static and does not move position. There is a User Interface (UI) at the top of the screen which the user uses for the visualization operations elaborated in Chapter 3.2. The biological data entities that were described in Chapter 2.3 are instantiated as game objects in Unity3D, where the pathway and gene nodes in the KGML are represented as child game object to the KEGG pathway image (Figure 3.5).

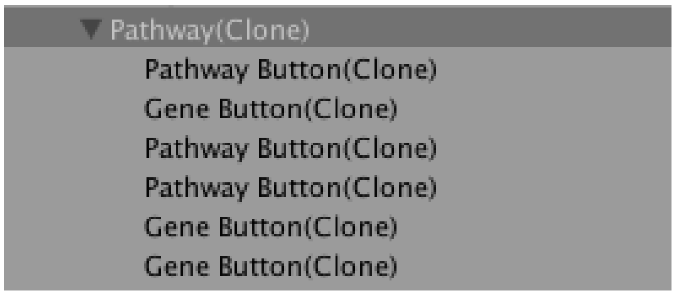


Figure 3.4: Pathway search results from the search term «Cancer».

3.2 Operations

We have implemented operations which we consider important for biological data exploration: i) search and visualize pathway, ii) search and highlight gene, iii) gene expression visualization, and iv) path discovery (connections between genes). Combined these cover a wide variety of visualization and exploration for researchers.

3.2.1 Pathway Search

A valuable functionality in a pathway exploration system is the ability to search for specific pathways. This is realized as a search field where users input search term such as «*cancer*». The search term is handled in the backend, Kvik framework [4]. It uses the search term with a fuzzy search in the KEGG databases. If there are any pathways that matches the inserted search term by a user, the matches are listed up as seen in Figure 3.5. The search results are presented with the title of the pathway. To not flood the screen, the result area is restricted to a portion of the screen with a scroll functionality. Clicking on a pathway from the list opens the pathway for visualizing.

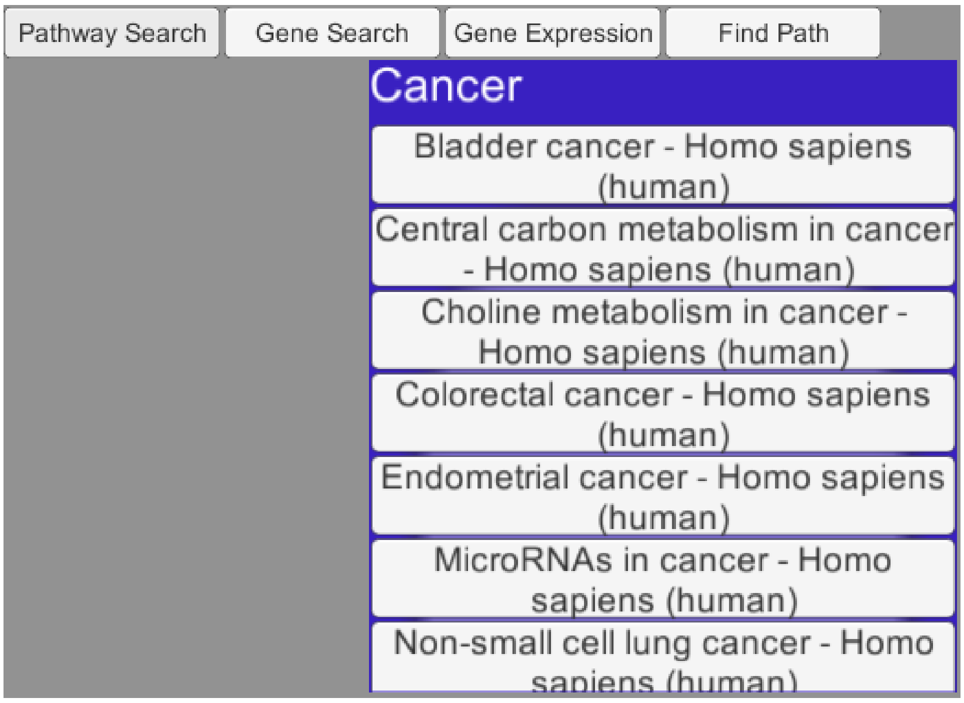

Pathway Search
Gene Search
Gene Expression
Find Path
Cancer
Bladder cancer - Homo sapiens (human)
Central carbon metabolism in cancer - Homo sapiens (human)
Choline metabolism in cancer - Homo sapiens (human)
Colorectal cancer - Homo sapiens (human)
Endometrial cancer - Homo sapiens (human)
MicroRNAs in cancer - Homo sapiens (human)
Non-small cell lung cancer - Homo sapiens (human)

Figure 3.5: Pathway search results to the search term «Cancer».

Pathway search is implemented in Unity3D as a text field where the user can insert text. Beneath this text field there is a restricted area that has the vertical scroll functionality. Every search term entry is sent to Kvik framework by using its REST Interface to the KEGG database. The kvik framework returns pathways that matches the search term and Freia is creating results that visualize the title and stores the id of the pathway in the result game object. For instance, the title «Pathways in cancer» has the pathway id «hsa05200».

When a result is clicked a pathway entity is instantiated and two new get requests are sent to Kvik framework. One request is for the pathway image and the other request is for the pathway KGML. Freia uses coroutines functions which waits for a response from the backend. The image returned from the backend is directly loaded into a Unity3D texture which is applied to the pathway entity. Parsing of the KGML is slower than the image retrieving and rendering (Chapter 4.3.1), so the pathway image is visible before the user can interact with its nodes. Every gene and pathway node that is present in the KGML is instantiated accordingly with its position, size, edges, name and id. The nodes are instantiated as transparent buttons as child game object to the pathway. This way they inherit the global space position of the pathway game object and will move and scale accordingly to the pathway.

3.2.2 Gene Search

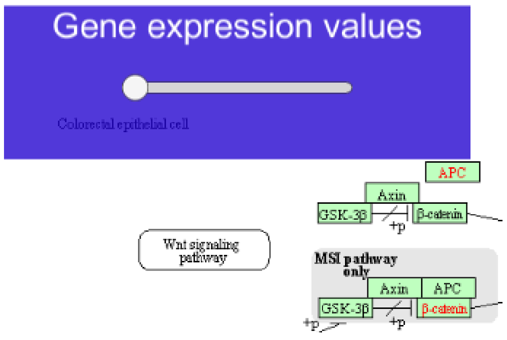
Another essential function is to search for specific genes. Freia features searching for specific genes within pathways, and highlights the results on the pathway visualization. Freia also features the possibility to highlight and remove highligted genes by clicking on the specific genes. There is no communication to Kvik framework since the gene search is on the active pathways

Every gene is overlaid on the pathway image as buttons. The button is a Unity3D class that has ability to listen to click events. When a gene is instantiated, the button is transparent and is therefore not visual. But when the button registers that it has been clicked on, it becomes visible, but transparent enough that the gene name is readable.

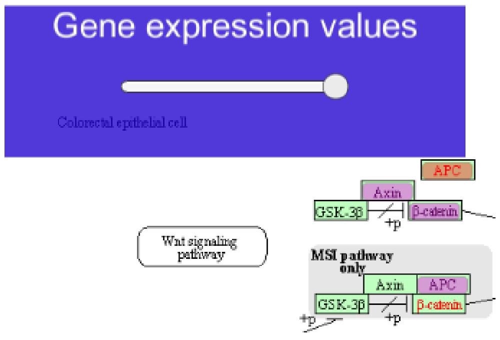
When a search for a specific gene is entered, Freia iterates through every pathway game object and every node in the pathway game object to check the entered gene name matches the name of the gene game object. If there is n number of pathways with m number of nodes, it has to iterate over n x m times. If the gene entity matches the entered search term, it is highlighted the same way as it was clicked.

3.2.3 Gene Expression Visualization

To get insight in how the processes are affected by gene expression levels in biological samples, it is common to overlay gene expression data on top of biological pathways. We retrieve the gene expression from a result of microarray technology from NOWAC through the Kvik Framework and is shown on top of the genes, see Figure 3.6. When the researchers apply gene expression data the genes will change in color if the genes are up-, or down regulated. Researchers can turn on and off the gene expression visualization by dragging the slider bar in Figure 3.6 on and off.



(a) Pathway with gene expression values not visible.



(b) Pathway with gene expression values visible.

Figure 3.6: Visualizing gene expression data on KEGG pathway. Slide bar operates as the on and off button.

The gene expression values that are retrieved from Kvik framework have in this experiment been simulated. This is done because the datasets used in this prototype is not relevant. A gene expression is a value from 0 to 1 and it does not matter where this value comes from. Although, the simulation data can be swapped to with real data from the NOWAC dataset with minor tweaks.

When the gene expression slider is active, every pathway game object is iterated and every node in the pathway game object. Every gene has a gene expression component which stores if the gene is up-, or down regulated. There might be cases where there are no gene expression data, and as seen in Figure 3.6 the genes color does not change. The visualization of the color is done by changing the transparency level with a light red or blue color.

The gene expression is a value from 0 to 1, but in this prototype it is either 0 or 1 and therefore it is distinguished between two colors. Since the colors is overlaid on the genes in the KEGG images which already has a color, it was cases where the text was to unreadable. The gene boxes could have been overwritten with a new game object, but the KGML had several cases were the name of the gene in the image and KGML were mismatched.

3.2.4 Path Discovery

There may be special interactions including a set of genes are involved, and this may be easier to discover if researchers can see how genes in a pathway are connected. Freia provides support for searching for paths between two genes in a pathway which is active in Freia. Figure 3.7 shows an illustration of a path between gene «araf» and gene «rps6ka5» in the Bladder Cancer (hsa05219) pathway. The path is highlighted by coloring the genes in the path blue.

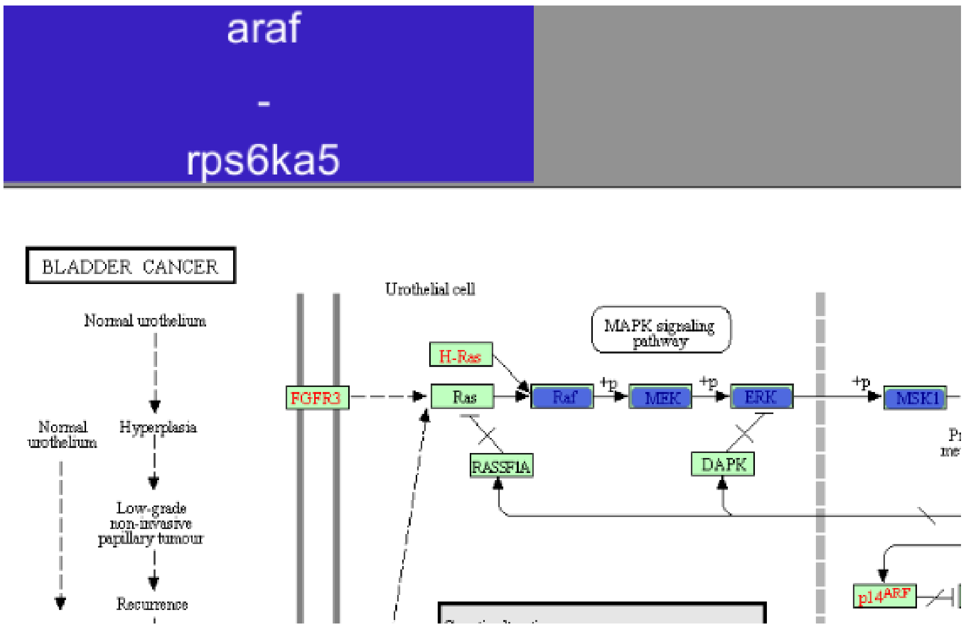


Figure 3.7: Path discovery in Freia.

When searching for a path between gene X and gene Y, Freia iterates over every pathway game object and thereafter every node in the pathway game object. It checks if gene X and gene Y to the gene node names in the pathway game object, and if both is present in the same pathway game object it can begin the recursion search. Since the links in the pathway images are directed graphs, the network path search only checks if there is a path from gene X to gene Y. Every gene game object has a link component which holds every other gene it links to. Freia uses first-depth recursion until gene Y is found or there are no more unvisited links. The path networking function visualize the path by changing the transparency level to a blue color.

4 Evaluation and discussion

For our Freia implementation, our main focus has been to investigating and demonstrate how we can map the biological data into the ECS model. However, performance and scalability is an important motivation for using game engines.

In this chapter, we evaluate Freia using micro-benchmarks by profiling with the built-in Unity3D profiler. We use the built-in profiler because of the lack of possibilities Unity provides when exporting performance data, such as memory usage. The profiler is a real-time debugging tool used by game developers in Unity3D. Further information about the built-in profiler capabilities will be discussed more in-depth in Chapter 4.2.

We will evaluate the following:

1. We measure Freias latency when loading a pathway. We will experiment with different pathway sizes in terms of number of nodes.

2. We will evaluate how «*smooth*» Freia is running. We measure smoothness using the frame rate. If the frame rate is lower than 30 FPS, viewer experience the visualization as slow and/or stuttering. We also want to evaluate scalability with respect to the number of visible pathways in the application.

3. We also evaluate how easy it is to implement visualization of biological

data with a game engine? This is a subjective experience from us, but Lines of Code (LOC) gives an indication of effect.

4.1 Experimental Setup

All of the experiments was run on a MacBook Pro running OS X Yosemite Version 10.10.5. The MacBook Pro has a 2,6 GHz Intel Core i5 processor, 8 GB 1600 MHz DDR3 memory and the Intel Iris 1536 MB graphic card. Unity3D version 1.5.1f1 Personal was used for this experiment. The experiments were run after rebooting the computer.

The experiments are performed by running the application within the editor and with the built-in Unity3D profiler. The profiler gives real-time feedback from the running application from Unity. We used the built-in profiler for running micro-benchmarks to measure performance. Since we ran the application within Unity instead of a standalone application, this may have interfered with some of the experimental data. We have three different experiments test input. How is the performance difference from one pathway, seven pathway or all the pathways[1](If it is possible to load all pathways).

The experiment was performed by selecting a random pathway out of the total amount of pathways that KEGG has. If we were not below our threshold, 30 FPS, we kept increasing the number of pathway until we found the roof.

4.2 Unity3D Profiler

As seen in Figure 4.2 the profiler has two main windows. The top window visualizes Central Processing Unit (CPU) and the memory usage. The visualization is categorized and is visualized with different colors. The rendering (green) and scripting (blue) categories are the ones that are affected most by our experiments and are therefore the most interesting. When selecting a frame inside the graph, a vertical white line will be shown as seen in the figure. Clicking a frame opens a new window with more detailed information, e.g. how many times a script is called and runtime relative to other scripts. It also distinguishes between child scripts, such as the «*Camera.Render*» script contains many child scripts. When selecting a script type it is highlighted in the top window. The top window is categorized in seven different categories, where our program logic falls under the «*scripts*» category.

1. In KEGG there are 203 human pathways at this moment

Note that the CPU usage shown in profiler is scaling accordingly for the highest peak in the current view. It does not imply that the highest peak uses 100 % of the CPU.

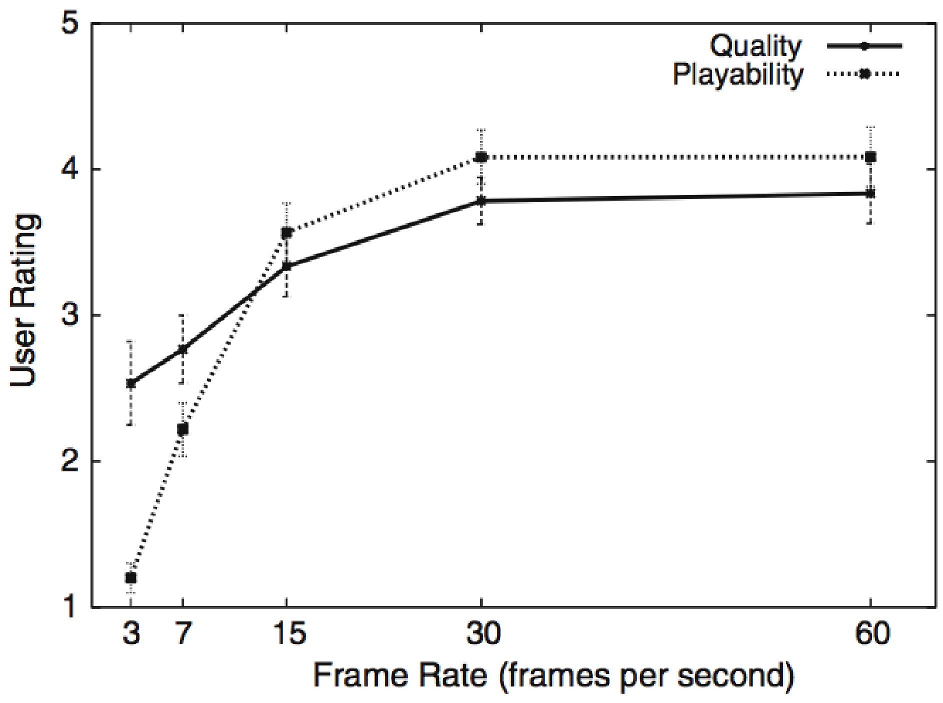
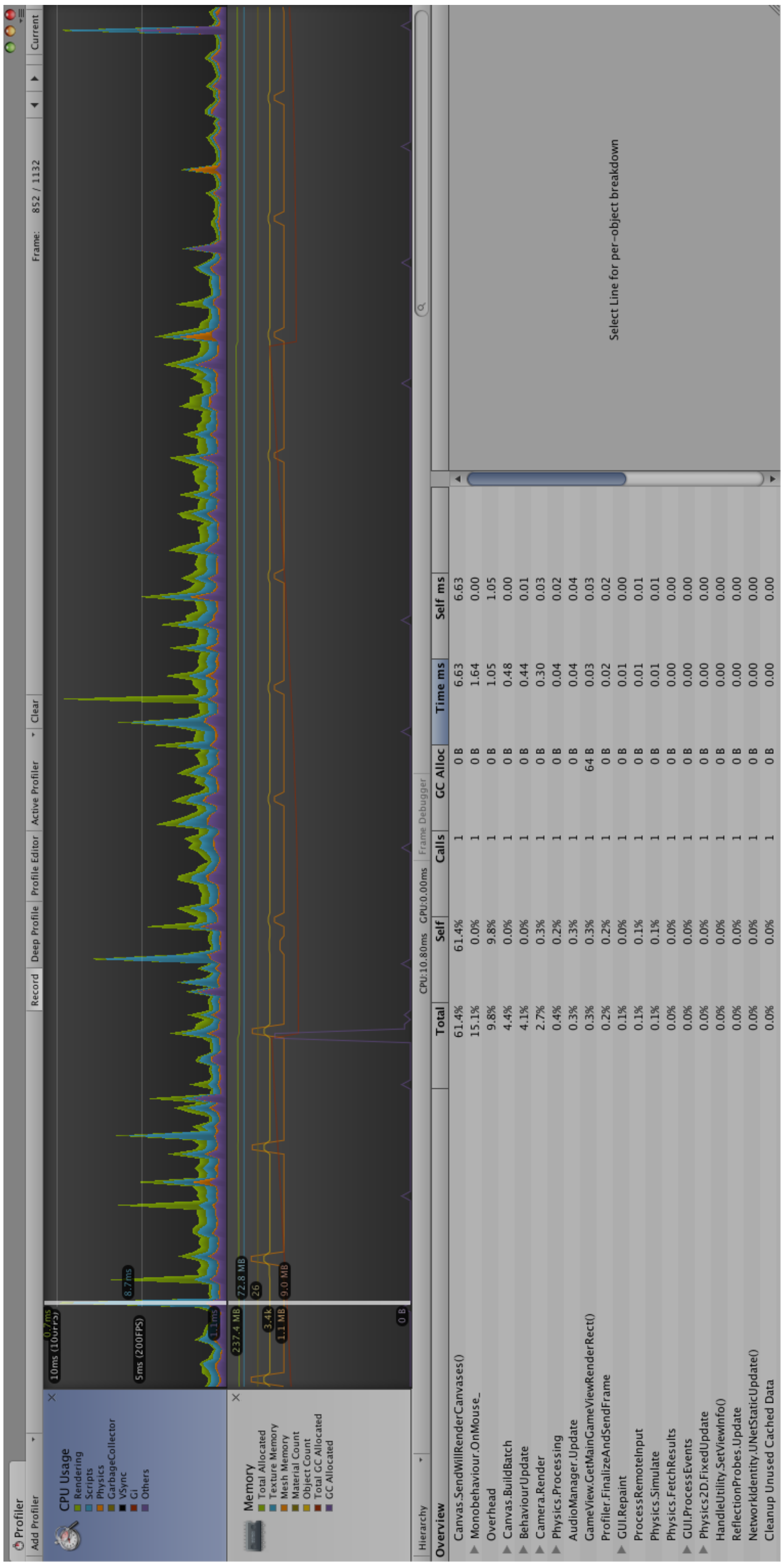


Figure 4.1: Effects of frame rate on user perception. Figure 19 from [8].

We use the results from [8] as target FPS. In [8], results from an experiment with one-hundred participants with a first-person shooter game. First-person shooter games are highly interactive and requires that the information on the screen is up to date at all times. As seen on Figure 4.1 there is little to no change in both quality and playability from 30 FPS and 60 FPS, but when the frame rate goes below 30 FPS both quality and playability declines. Freia should at least 30 FPS in order to achieve «*smooth*» performance.



Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
Canvas.SendWillRenderCanvases()	61.4%	61.4%	1	0 B	6.63	6.63
▶ Monobehaviour.OnMouse_	15.1%	0.0%	1	0 B	1.64	0.00
Overhead	9.8%	9.8%	1	0 B	1.05	1.05
▶ Canvas.BuildBatch	4.4%	0.0%	1	0 B	0.48	0.00
▶ BehaviourUpdate	4.1%	0.0%	1	0 B	0.44	0.01
▶ Camera.Render	2.7%	0.3%	1	0 B	0.30	0.03
▶ Physics.Processing	0.4%	0.2%	1	0 B	0.04	0.02
AudioManager.Update	0.3%	0.3%	1	0 B	0.04	0.04
GameView.GetMainGameViewRenderRect()	0.3%	0.3%	1	64 B	0.03	0.03
Profiler.FinalizeAndSendFrame	0.2%	0.2%	1	0 B	0.02	0.02
▶ GUI.Repaint	0.1%	0.0%	1	0 B	0.01	0.00
ProcessRemoteInput	0.1%	0.1%	1	0 B	0.01	0.01
Physics.Simulate	0.1%	0.1%	1	0 B	0.01	0.01
Physics.FetchResults	0.0%	0.0%	1	0 B	0.00	0.00
▶ GUI.ProcessEvents	0.0%	0.0%	1	0 B	0.00	0.00
▶ Physics2D.FixedUpdate	0.0%	0.0%	1	0 B	0.00	0.00
HandleUtility.SetViewInfo()	0.0%	0.0%	1	0 B	0.00	0.00
ReflectionProbes.Update	0.0%	0.0%	1	0 B	0.00	0.00
NetworkIdentity.UNetStaticUpdate()	0.0%	0.0%	1	0 B	0.00	0.00
Cleanup Unused Cached Data	0.0%	0.0%	1	0 B	0.00	0.00

Figure 4.2: Builtin profiler in Unity3D.

4.3 Experiments

4.3.1 Load Pathways

We compare the result from Freia with Kvik Pathways [7] therefore use the same four pathways as [7] (Table 4.1).

Loading a pathway requires both the pathway KGML and image from Kvik framework. The latency is measured from the user click on a pathway result from the search field, until it is rendered with all the overlay node as seen in Figure 3.2.

Table 4.1: Pathways used to evaluate Freia.

Id	Name	Number of nodes
hsa04630	Jak-Stat signaling pathway	35
hsa04915	Estrogen signaling pathway	74
hsa04151	PI3K-Akt signaling pathway	120
hsa05200	Pathway in cancer	267

Table 4.2 shows the result for loading different pathways. The results shows that the load time increases as the number of nodes increases, but for the average pathway the load time would be 1.5 second. Pathways with a larger number of nodes uses almost a second longer than the average load time. The variance in load time is small and not notable for the users eye.

Table 4.2: Time to load pathway visualization.

Id	Average	Standard Deviation
hsa04630	0.97s	0.10s
hsa04915	1.27s	0.05s
hsa04151	1.60s	0.05s
hsa05200	2.24s	0.01s

[9] reports that there are many different definitions of response time. In our case the «*System, can you do work for me?*» definition from [9] is the most applicable. This definition states that the acknowledgement should be within two seconds. As seen in Table 4.2 the largest pathway hsa05200 (Pathways in cancer) does not satisfy this response time definition. As the loading time is increasing as the number of nodes is increasing in the pathways, it indicated that it is the processing of the KGML is the most CPU expensive. The parsing of the response from the Kvik framework and instantiation of new game objects is the cause of slow processing. Requesting and rendering only the image takes approximately 20 milliseconds, it does not matter if it has few or many nodes.

This is exploited in order to achieve user perceived latency by visualizing the image before all the KGML data have been processed. The user is most likely to position and/or scale the image before starting to access the KGML data.

Comparing Freia (Table 4.2) with Kvik Pathways (Table 4.3), the latency of all pathways are higher on Freia. We assume that it takes longer time because of the instantiation of new game objects within Unity3D. The large hsa05200 pathway has 267 nodes which implies that Unity3D has to instantiate one game object for the pathway itself and then accordingly one game object per node. Instantiating and destroying game objects is inefficient when it occurs frequently [10].

Freia is more consistent when it comes to standard deviation, the larger the pathway the less deviation was registered. For the small pathway the deviation is 0.1 second which is not notable for the eye when requesting the same pathway image over and over. Kvik Pathways has a larger deviation of response time and they suspect that it is the garbage collector in Firefox that is responsible for it [7]. This may be applicable if we deployed Freia as an web application, and it would therefore maybe affect Freia the same as it did with Kvik Pathways.

Table 4.3: Time to load pathway visualization from Kvik Pathways[7].

Id	Geometric mean	Geometric SD
hsa04630	0.20s	1.30s
hsa04915	0.34s	1.20s
hsa04151	0.62s	1.26s
hsa05200	1.00s	1.48s

4.3.2 Interaction with Pathways

We use the profiler while doing the operations described in the case study in Chapter 3.2.

When we visualize different number of pathways at the same time. We want to find, if there is any, upper bound on number of visible pathways. As [8] states that there is a drop in quality and playability in first person shooter games as the FPS falls below 30 FPS. We checked the FPS performance of Freia when an image was dragged and with applying gene expression data on every opened pathway. These operations were picked because they are the two most visual operations, and therefore they are easier for the eye to registries that it is not running smooth.

We do the following steps:

1. **Open** Freia.

2. **Enter** the search term, «*cancer*», in the search field that is open when Freia is started up and press enter.

3. **Click** on the result.

4. **Drag and scroll** the image.

5. **Press** the «*Gene Expression*» box and drag the slider to the right. The gene boxes which has data will now show up regulated (red) or down regulated (blue).

6. **Press** the «*Find Path*» box and enter a start gene and end gene. The path found is shown.

We created a C# script that could load specified number of pathways to help us do experiments with a larger number of pathways. This same script were also used to calculate the latency load for visualization of the single pathways.

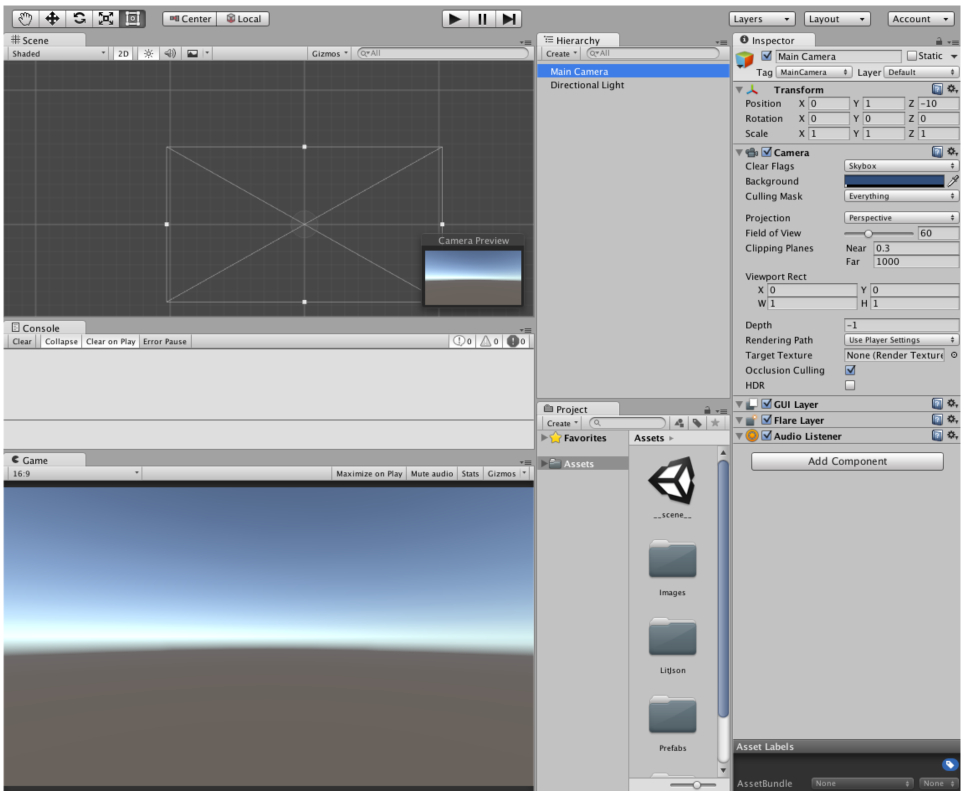
The experiment for one pathway we can that both dragging (Figure A.1) and gene expression (Figure A.2) operations satisfy our over 30 FPS requirement. By comparing Figure A.1 and Figure A.2 it is clear that applying gene expression on every gene is more CPU expensive and it affects the FPS way more. Figure 2 has a high blue peak which indicates that it is the scripting that is expensive, and as seen in the overview window below it is shown that it is the event system update function that is eating CPU cycles. As described in Chapter 3.2.3 the gene expression is coloring a transition from transparent to a noticeable color and back, depending on which way the slider is. If this was only implemented as an on or off it would maybe be more efficient for the CPU.

One pathway with gene expression (Figure A.2) operation ate a lot of CPU and gave Freia a frame rate of 60 FPS in comparison with a frame rate on 120 FPS with drag operation. The gene expression operation is coloring every gene that have data on it, and this is on every pathway that is open. The dragging operation is only done on one pathway image at a time, so this operation does not scale with the number of pathways as the gene expression operation does.

Seven pathways are the expected pathway work size that researchers would work with in order to gain a proper overview. Dragging an image with seven pathways open has a frame rate of 110 FPS average as shown in Figure A.3.

The frame rate has dropped from 120 FPS in one pathway to 110 FPS and it is acceptable. With seven pathways open and with gene expression operation it barely satisfies the frame rate requirement as seen in Figure A.4. In Figure A.5 the frame rate has been cut in half and it is clearly violating the frame rate requirement.

The drag operation can handle many more pathways than the gene expression operation, it was not violating the requirement until it had 50 pathways. In Figure A.6 it is clear that the rendering is consuming more CPU cycles than the script part. Freia can almost handle 100 pathways (Figure A.7) running simultaneously, but this is with no interactions. Freia serves no use if it can not be interacted with, but as earlier noted it is expected that the average workload in number of pathways is seven, which clearly satisfy the requirement. 50 and 100 simultaneously running pathways is not expected at all, but is used for measurement to see what Freia can handle.



Main Camera Static
Tag MainCamera Layer Default

Transform
Position X 0 Y 1 Z -10
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Camera
Clear Flags Skybox
Background
Culling Mask Everything
Projection Perspective
Field of View 60
Clipping Planes Near 0.3 Far 1000
Viewport Rect X 0 Y 0 W 1 H 1
Depth -1
Rendering Path Use Player Settings
Target Texture None (Render Texture)
Occlusion Culling ✓
HDR

GUI Layer
Flare Layer
Audio Listener

Figure 4.3: Screenshot of Unity3D with several developer windows open.

4.3.3 Using a Game Engine for Visualization of Biological Data

Game developers have throughout many years gathered routines and frameworks which can be reused in a variety of games. These routines and frameworks have merged over time to game engines, which is a platform dedicated for game development. We will in this chapter present how it is to develop with a game engine and how easy is it to develop a visualization of biological data with the game engine.

Coming from a scripting/programming environment, the graphical overview and utilities/functions were overwhelming at the beginning of the project. As seen in Figure 4.3 there are many windows that provide with project overview in their way. The scene window provides with overview of the game objects in the global space of Unity3D. The consol window gives feedback on errors and can be used to debug. The game window is what the user sees when the application runs, aspect ratio can be specified and change on the go. In the hierarchy window is every game object in the game listed up. A game object can be created by just right clicking in this window and the developer can choose by selecting several pre defined game objects such as UI or audio game object or it can be an empty game object. In the project window holds all the files that is used in this Unity3D project, e.g. images, scripts etc. The inspector window shows all the components that is bound to the selected game object, the selected game object can either be in the hierarchy window or in the project window.

Unity3D shines when it comes to tweaking. By exposing attributes in the script components, these attributes become visible in the Unity3D editor. When a game object is selected its components is exposed in the inspector window. In Figure 4.3 the main camera is selected and its components is shown in the inspector window. There are 5 components shown in the inspector window, and in each component the public attributes are shown. Here it is possible to adjust the attributes instead of accessing the script. Unity3D adjust the variable input to the declared type, e.g. check boxes for boolean attributes. Even a researcher could tweak variables in the editor of Unity3D, as it requires as much effort as typing in the search field for pathways.

A game engine is an interactive game development tool. Unity3D provides with an editor window which includes an interface to position objects and assign components to objects. E.g. an implemented script can be dragged from the project window onto an object. This reduces the amount of LOC that has to be written, and the scripting part can be hidden from non coders such as graphic artists.

The mapping from the biological data to the ECS model and how to do this in Unity3D was the most time consuming in this project. The visualization did not go without a hassle. Unity3D operates with three different coordinate systems; world point system, screen point system and viewport point system. When objects coordinates need to be relative to each other, you may need to convert from one coordinate system to another. World point system is how the objects are relative to the world space in the application, it is the absolute XYZ coordinates of the objects. Viewport point system and screen point system represent the same area on the screen and both are represented in 2D, but they have different coordinate system. The viewport space is where the camera renders and is typically used for UI elements. All inputs from a mouse or touch is received in the screen space, and these have to be mapped to viewport space for the UI elements. Including to these three coordinate systems there are also a local space as opposed to the world space. Local space is the coordinate relative to another object, and this was something we could exploit when attaching the gene and pathway nodes in the pathway game object. Unity3D is a game engine full of features from visualization support to controller support, but thanks to the Unity3D documentation [11] and the Unity3D use community [12] which is great when first getting acquainted with Unity3D and for further experience.

In our case we have 692 LOC where the code is consisting of initializing game objects, aligning game objects in the coordinate system and storing, get requests, parsing responses and storing attributes. Comparing our LOC to the demographic data shown by [13] Freia has less LOC than a simple iPhone game application. The demographic data is displaying the LOC different application has. The simple iPhone game application in the graph [13] presents may include external libraries and so on, but it is an indicator for complexity of the application. With LOC and out subjective experience we conclude that visualization of biological data with a game engine may be accomplished without great difficulty.

Memory

We did not specify any requirement for memory usage, but as the experiment were done we noticed some trends. For every pathway we instantiated it used approximately 22 MB, regardless if the pathway had few or many nodes. Every node is instantiated, so we would expect that a pathway that has 260 nodes would use more memory than a pathway that has 40 nodes. This indicates that the structure that is used is maybe bad memory wise and need further reconstruction.

4.4 From 2D to 3D Visualization

Our end users would like to have the KEGG pathways and the biological data represented as 3D. We see the potential to further develop Freia to be a 3D application where the researchers may explore in a way richer way. The only notion of 3D at the current state of Freia is the scaling of the pathways images.

Developers can be convert the 2D visualization to 3D visualization with some small and some bigger changes. The components to the camera have specifically been set to behave as an 2D camera, so there is no notion of 3D. The pathway entitiesuse the built-in 2D colliders which is under the physics 2D engine. These may have to be changed for 3D to be active in a 3D environment, but since the pathway images are 2D it may be a possibility for not having to do that. Instead of scaling the image in a 3D application we could use the z-coordinate to make the same effect.

Before any 3D visualization can be made of the biological data, it is important to understand how everything is connected. As the KEGG database provide the KGML for only external use, the KGML is missing information about the pathway images, e.g. added annotation such as genes that are inside of a cell membrane. There might be a need for completely KGML that is describing everything that is in the pathway image, or there has to be some sort of tool that can recognize the objects in the pathway images as in [14]. Creating a 3D visualization with only KEGG pathway images and KGML can be done as the 2D visualization, but the presentation of the KEGG pathway images can be confusing if they are not aligned properly. Pathways which are related to each other, or have interactions with each other should be aligned close to each other in the 3D space. This information is not known from the KGML and it requires therefore further investigation.

4.4.1 Pathway connectivity in KEGG

Since there are not any databases that have 3D representation of biological processes, the solution to 3D visualization of biological processes is to create a 3D world with KEGG pathways. We did some initial experiment for how connected the KEGG pathways for human diseases are. We had some question regarding how everything were connected and how we should interpret this information and use this for our advantage when mapping this biologic data. This experiment was done by using D3[2] library where it received the requests from the Kvik framework.

2. d3js.org

In Figure 3.7 every node is a pathway and there is an edge between connected pathways. We discovered that even though there is a connection between two pathways, it does not mean that there is a connection from both pathways to each other, so the graph is a directed-graph. For further connectivity visualization such as 3D visualization of the pathways, this has to be indicated somehow. Gene expressions in pathway X may effect gene expressions in pathway Y, but not the other way around.

In Figure 3.7 the two clusters of nodes show two groups with a strong relationship inside them. There are also a group of nodes that are not connected to any other node. This raises the question to how to explore to these pathways when clicking through the pathways scheme. Even though pathways may not be connected, these may share genes or compounds which affects biological processes. We should therefore have a network path search that would show every pathway that has the same path between two given genes.

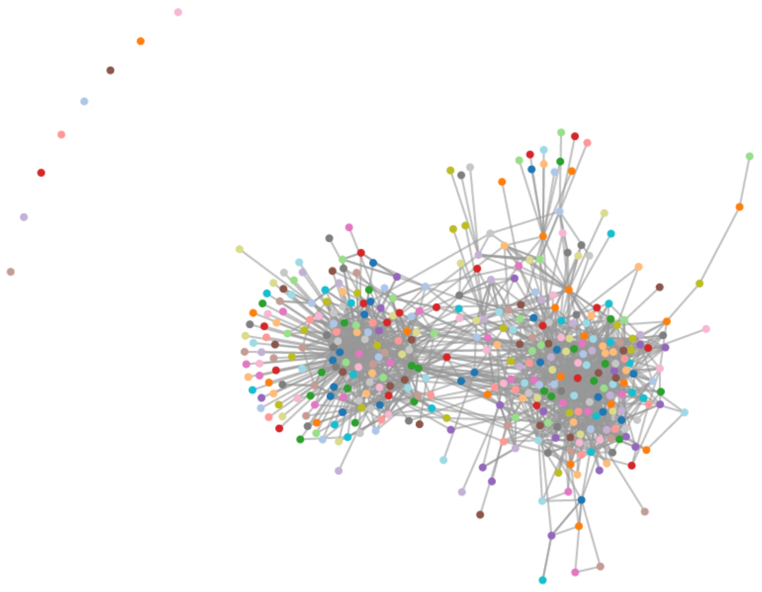


Figure 4.4: Pathway connection clustering. Pathways are represented as nodes and pathway connection is the edges between the pathways.

5 Related Work

While there exist several tools and application for visualizing and exploring biological data, the biology projects often requires custom built solutions for their specific problem/data set.

This chapter will give a short introduction to other visualization systems.

5.1 Kvik

Kvik is framework for developing applications for exploration of biological data. Kvik provides a simple interface to systems for executing statistical analyses and retrieving information from meta-databases, such as KEGG. [4] Kvik is designed as a modular framework that allows developers to build light-weight applications that interact with powerful compute and storage resources. Kvik Pathways is an application that is built using the Kvik framework. Kvik Pathways is a web application that allows researchers explore gene expression in the context of KEGG pathways. [4] Similar to Freia, Kvik Pathways visualizes biological pathways by overlaying nodes on top of the static pathway image from the KEGG database. Figure 5.1 shows the user interface in Kvik Pathway.

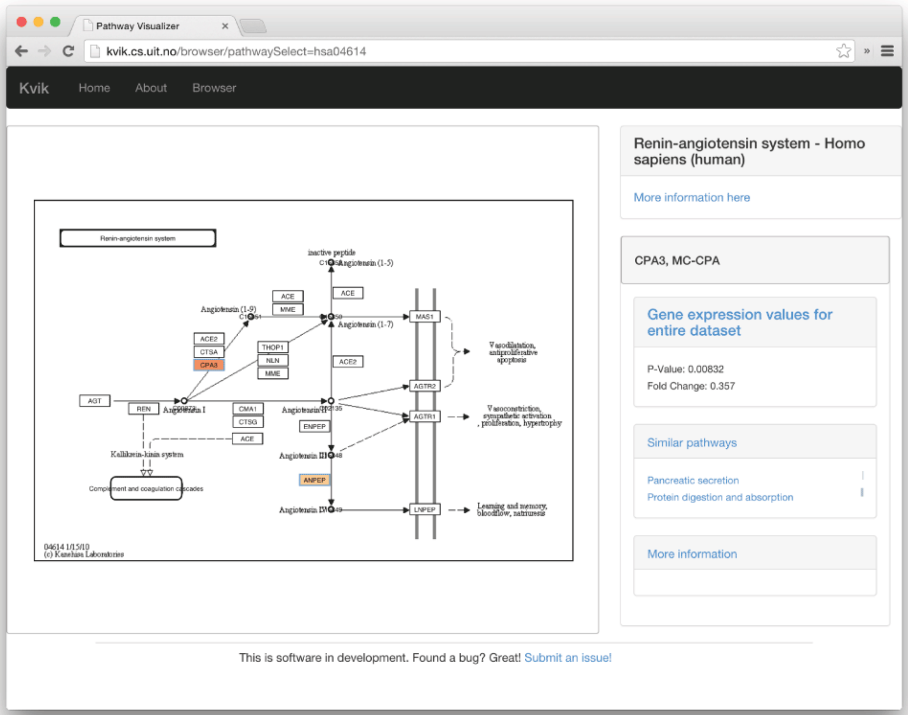

Pathway Visualizer
kvik.cs.uit.no/browser/pathwaySelect=hsa04614
Kvik
Home
About
Browser
Renin-angiotensin system - Homo sapiens (human)
More information here
CPA3, MC-CPA
Gene expression values for entire dataset
P-Value: 0.00832
Fold Change: 0.357
Similar pathways
Pancreatic secretion
Protein digestion and absorption
More information
This is software in development. Found a bug? Great! Submit an issue!

Figure 5.1: Screenshot of the renin-angiotensin pathway (KEGG pathway id hsa04614) in Kvik Pathways.

5.2 KEGG

KEGG is an integrated database resource consisting of a collection of 16 databases which again are categorized into four systems [15]. It is used for understanding high-level functions and utilities of the biological systems. One of their main database is the *pathway* database which consists of a collection of manually drawn pathway maps that represents their knowledge on the molecular interaction and reaction networks. KEGG is freely available through its website[1] or through the GenomeNet mirror website[2] where users may retrieve the images in PNG format and exchange format KGML.

1. kegg.jp
2. genome.jp/kegg

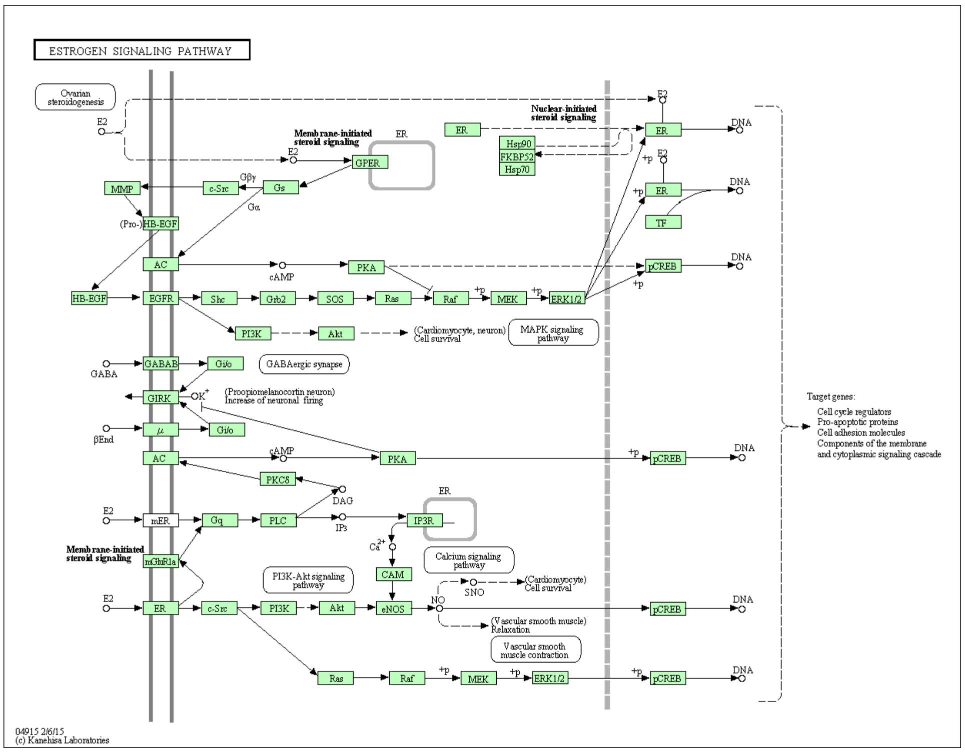

ESTROGEN SIGNALING PATHWAY
Ovarian steroidogenesis
Membrane-initiated steroid signaling
Nuclear-initiated steroid signaling
Target genes:
Cell cycle regulators
Pro-apoptotic proteins
Cell adhesion molecules
Components of the membrane and cytoplasmic signaling cascade
04915 2/6/15
(c) Kanehisa Laboratories

Figure 5.2: KEGG pathway for estrogen signaling, *rest.kegg.jp/get/hsa04915/image*.

The resource that KEGG is freely available, but if the users of the KEGG database want push notification or access to download the whole database, users may subscribe for a File Transfer Protocol (FTP) service to gain access.

5.3 BioCarta

BioCarta is an interactive online resource targeted to the life science research community [16]. It is similar to KEGG in the context that users may search for pathways and click on genes to view additional information. But the pathway section in BioCarta, the gene interactions are presented as dynamical graphical models (Graphics Interchange Format (GIF)). Figure 5.3 shows the Wnt signaling pathway from BioCarta.

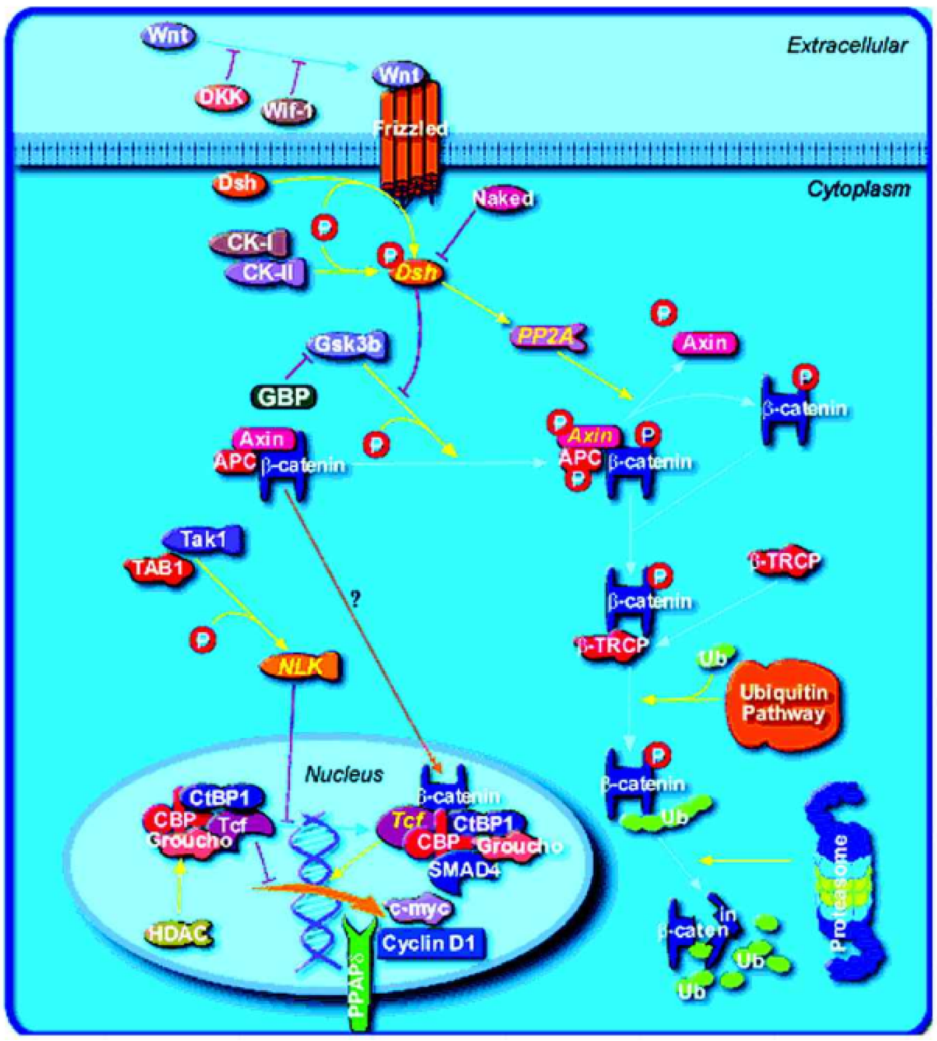


Figure 5.3: Screenshot of the Wnt signaling pathway in BioCarta. Figure taken from [16].

The pathways and data were not updated any more, but the pathways are still available at the Cancer Genome Anatomy Project[3].

5.4 KEGGViewer

KEGGViewer[17] is a BioJS[3] component to visualize KEGG pathways. It uses the KGML representations of pathways from the KEGG REST API to

3. cgap.nci.nih.gov/Pathways/BioCarta_Pathways

build pathways, and visualizes them in a web browser using Javascript library Cytoscape[18]. Since KEGGViewer only uses the KGML representation to generate the visualizations, they are missing added annotation as well nodes and edges.

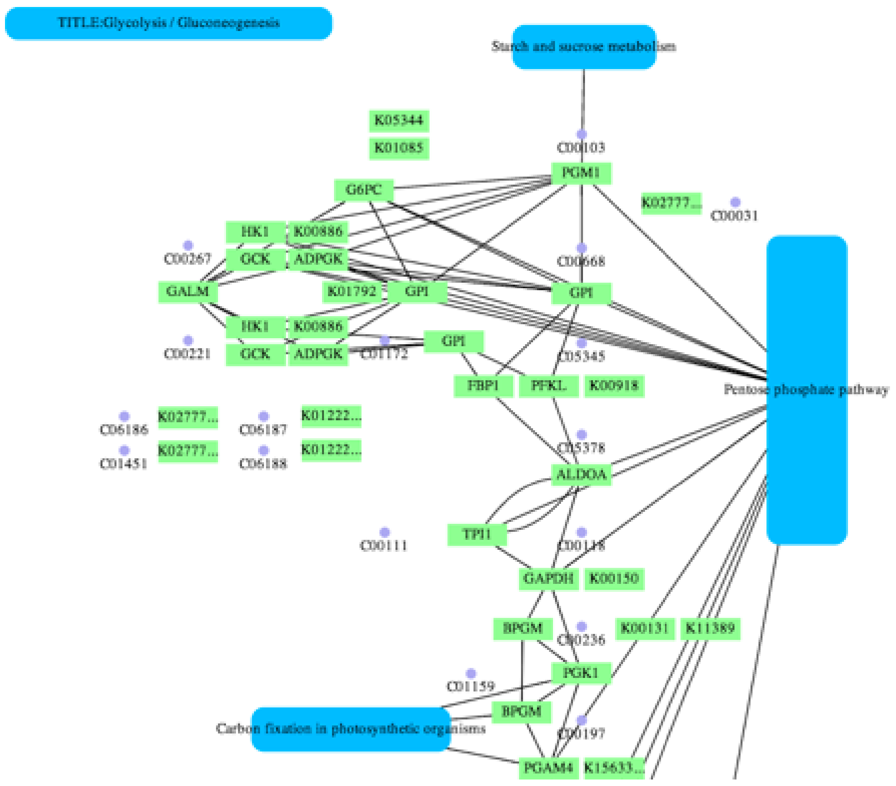


Figure 5.4: Overview of the KEGGViewer component[17].

5.5 Caleydo

Caleydo is a visualization system that addresses and supports two work flows, pathway-centric approach and analysis of gene expression data [19]. It supports two pathway databases, KEGG and BioCarta, which consist together of 600 pathways. Uses traditional multiple views for viewing large datasets or highly interactive visualizations, also supports 2.5D technique in order to support a seamless navigation of multiple pathways which simultaneously links to the expression of the contained genes. Figure 5.5 shows the bucked view with integration of both the KEGG pathways and BioCarta pathways.

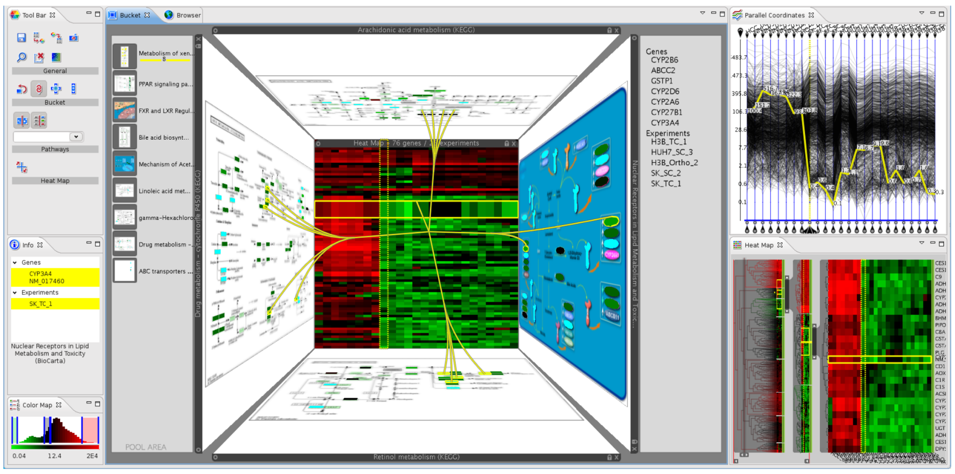


Figure 5.5: Caleydo visual analysis framework for gene expression data in its biological context. Figure taken from [19]

Caleydo have in the later years divided its task into smaller projects and today it consists of 7 projects [20].

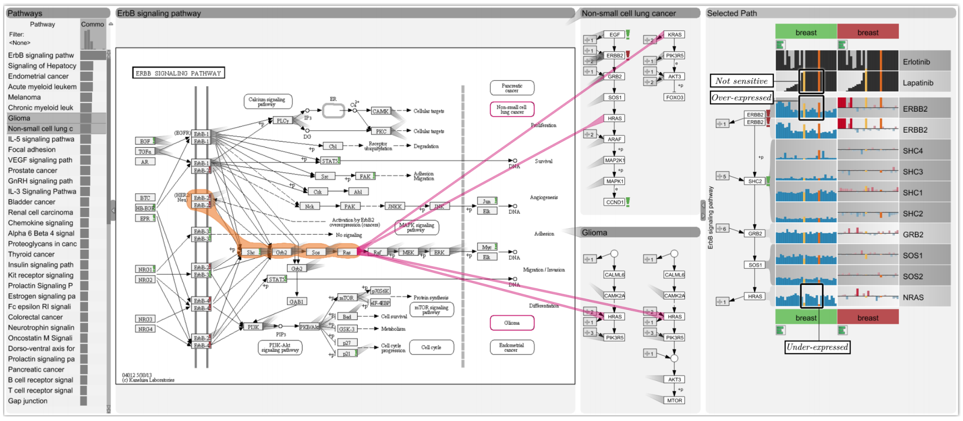


Figure 5.6: Entourage showing the Glioma pathway in detail and contextual information of multiple related pathways.

5.6 Entourage

Entourage is a visualization technique that provides contextual information when visualizing multiple related pathways [21]. It uses a single focus pathway for main interaction and exploration, and visualizes only what is important to researchers from other related pathways. Entourage visualizes subsets of related pathways to give context information about the location of user selected genes within related pathways. Entourage is using previous existing techniques such as enRoute [22] technique in order to visualize experimental data and Bubble Sets [23] in order to highlight the selected nodes and the the route for these nodes within a pathway.

5.7 Dynamic Exploration

Dynamic exploration draws pathway and connects them to each other in an overlapping way [24]. Pathway links within another pathway is drawn and all the links is shown as seen in Figure 5.7. The user may explore through all of the connected pathways whenever there is a connection to another pathway. The pathways are drawn and connected with the KGML, which causes lack information such as the annotated data as well as nodes and edges.

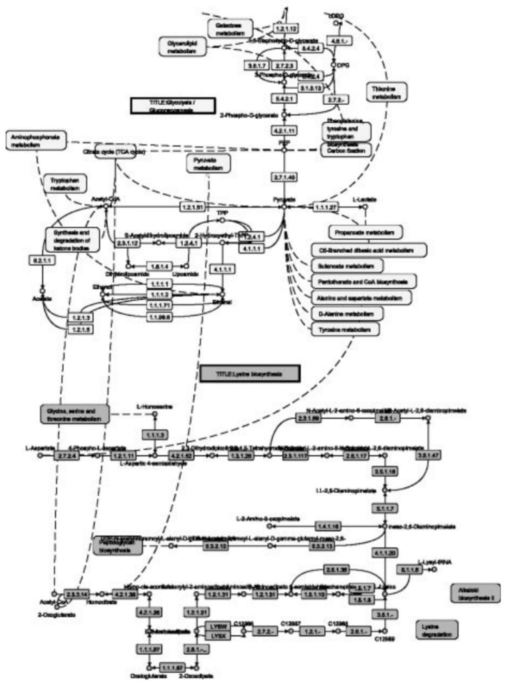


Figure 5.7: Dynamic exploration of two pathways with connected edges. Figure 2 from [24].

5.8 UnityMol

UnityMol is a prototype for displaying biological network and molecular visualization for research or educational use [25]. The UnityMol prototype is built with Unity3D and is exploring the possibility for quick developments for educational or scientific purpose. UnityMol visualization is compared to an original cytoscape[18] visualization in Figure 5.8. UnityMol is released in two versions, a stand-alone application and a web applet which runs on top of Unity3D web-plugin. UnityMol is using the built-in graphical primitives in Unity3D to create 3D content with spheres and point-sprite particles. Our approach does not have the 3D representation of the biological network as their approach is based on.

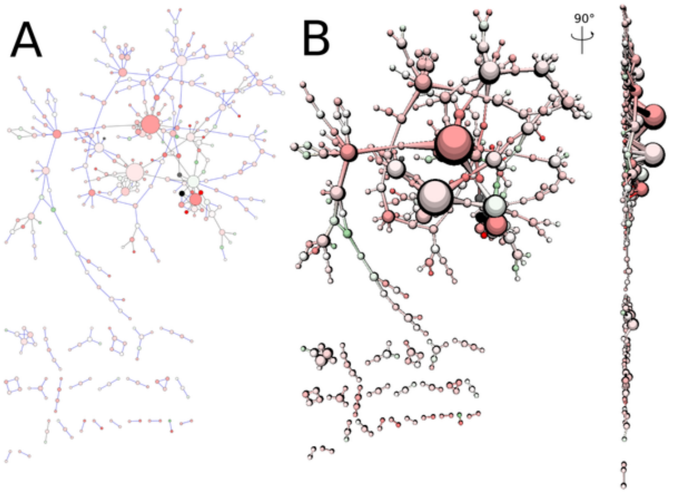


Figure 5.8: Comparing a cytoscape visualization (A) to a UnityMol visualization (B). Figure 6 from [25].

6 Conclusion

We have presented to our knowledge the first approach for mapping biological data to game engine structures. Our approach allows biological visualization to utilize the efficiency brought by the game engine structures. We implemented Freia to demonstrate the approach.

Users may search for specific domain knowledge, such as pathways, genes and paths between genes. By using a game engine, we can deploy to a multitude of devices and develop in substantially less time and with less developer effort. This raises huge potentialities for quick development and our users may also contribute by using the user friendly interface.

Our approach to visualize and explore biological data is by using a game engine. We use familiar representation of biological processes (KEGG pathway) and operations that utilizes the search for patterns regarding cancer development. Such operations must include multiple views of biological processes, search for biological processes, search within biological processes and search for connections in the biological processes. This have been realized as pathway search, gene search, gene expression visualization and path discovery.

Although we apply the biological data to a specific data type (gene expression) and representation of biological processes (KEGG), within the game engine, the approach is more general. By using the ECS model, game objects in the game engine can have components that represent different data types and biological knowledge. A pathway game object can for example get biological knowledge from other databases such as Reactome, rather than KEGG. This allows researchers to combine different image representation of biological processes, and this may help the researchers with exploring of patterns that may be related to the development of cancer.

Exploration of biological processes are essential for researchers to find patterns and understanding hypothesis about the development of cancer. We have shown Freias functionality which support exploration of biological processes, visualization of patient's gene expressions, and path discovery between genes. We believe Freia is important for visualizing, exploring and discovering biological processes such as the visual representation by KEGG. Finally, the possibility by the game engine to automatically deploy on major platforms and devices, provides this approach to reach out to a broad audience.

Freia is released as an open-source project hosted at github: `github.com/knudah/freia`.

The functionality provided by Freia can be viewed at youtube: `youtu.be/22XmfSY0w08`.

6.1 Future Work

Freia fulfills our initial requirements and has demonstrated the advantages and possibilities. We have mentioned some directions for improvements earlier and we summarize them here:

Pre caching Freia is sending every pathway search to the Kvik framework. We believe that the meta data of the human pathways would not be heavy for Freia to store and this would reduce the network traffic. Although the network traffic is not big, but having the meta data about pathways in Freia gives further possibilities in other functionalities.

Path networking The researchers were quite excited about the ability to search for paths between two genes. This search work on genes, so if there is a pathway or a compound between, the path would not be detected. There are not every pathway that has the compound links in the KGML.

The path search only search for paths in the opened pathways. If the meta data about the pathways were stored in a data structure in Freia, it could be expanded to search for the path between gene X and Y in all of the pathways.

3D visualization Our researchers would like the visualization in 3D. There are not any 3D models database that represents biological processes in the scale of what KEGG provides. The Unity3D engine has the possibility to create Freia from 2D to 3D with minor changes. In Chapter 4.4.1 we did some experiments how the connectivity was in the KEGG pathways. We noticed there were some clustering of the pathways which were very connected to each other. When the pathways are this connected, it must be a smart solution to how to explore these pathways in a intuitive way.

Memory utilization We noticed that every instantiation of a new pathway (game object) in Unity3D used around 22 MB. Unity3D themselves claims that instantiation is inefficient [10]. For more effective ways there could be used some pooling systems, which holds x number of game objects in memory at all time and does not instantiate and destroy game objects.

ECS model Our implementation of the ECS system is not efficient hence the lack of system implementation. The entities and components are implemented, but the functionality such as the drag operation that should be handled by the movement system is not implemented. These have been implemented inside the components. When the system is implemented, it is room for better memory utilization, aligning components of the same type in memory.

Bibliography

[1] Rebecca L Siegel, Kimberly D Miller, and Ahmedin Jemal. Cancer statistics, 2015. *CA: a cancer journal for clinicians*, 65(1):5–29, 2015.

[2] Eiliv Lund, Vanessa Dumeaux, Tonje Braaten, Anette Hjartåker, Dagrun Engeset, Guri Skeie, and Merethe Kumle. Cohort profile: the norwegian women and cancer study—nowac—kvinner og kreft. *International journal of epidemiology*, 37(1):36–41, 2008.

[3] John Gómez, Leyla J García, Gustavo A Salazar, Jose Villaveces, Swanand Gore, Alexander García, Maria J Martín, Guillaume Launay, Rafael Alcántara, Noemi Del-Toro, Marine Dumousseau, Sandra Orchard, Sameer Velankar, Henning Hermjakob, Chenggong Zong, Peipei Ping, Manuel Corpas, and Rafael C Jiménez. BioJS: an open source JavaScript framework for biological data visualization. *Bioinformatics (Oxford, England)*, 29:1103–4, 2013.

[4] Bjørn Fjukstad, Karina Standahl Olsen, Mie Jareid, Eiliv Lund, and Lars Ailo Bongo. Kvik: three-tier data exploration tools for flexible analysis of genomic data in epidemiological studies [version 1; referees: 2 approved with reservations]. *F1000Research 2015, 4:81 (doi: 10.12688/f1000research.6238.1)*, 4, 2015.

[5] Scott Bilas. A data-driven game object system. In *Game Developers Conference Proceedings*, 2002.

[6] Karina S Olsen. *Blood gene expression, lifestyle and diet - The Norwegian Women and Cancer Post-genome Cohort*. Doctoral thesis, University of Tromsø, 2013.

[7] Bjørn Fjukstad. Kvik: Interactive exploration of genomic data from the nowac postgenome biobank. 2014.

[8] Kajal T Claypool and Mark Claypool. On frame rate and player performance in first person shooter games. *Multimedia systems*, 13(1):3–17,

2007.

[9] Robert B Miller. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 267–277. ACM, 1968.

[10] Unity3D. Object pooling. `https://unity3d.com/learn/tutorials/modules/beginner/live-training-archive/object-pooling`. [Online; accessed 05-October-2015].

[11] Unity. Unity manual. `docs.unity3d.com/Manual/index.html`. [Online; accessed 05-November-2015].

[12] Unity. Unity community. `unity3d.com/community`. [Online; accessed 05-November-2015].

[13] Information is Beautiful. Codebases - millions of lines of code. `informationisbeautiful.net/visualizations/million-lines-of-code`. [Online; accessed 05-November-2015].

[14] Kenneth Knudsen. Amdex: automated meta-data extraction from kegg pathways. *Bachelor Thesis, Dept. of Computer Science, University of Tromsø*, 2014.

[15] Minoru Kanehisa, Yoko Sato, Masayuki Kawashima, Miho Furumichi, and Mao Tanabe. Kegg as a reference resource for gene and protein annotation. *Nucleic acids research*, page gkv1070, 2015.

[16] Darryl Nishimura. Biocarta. *Biotech Software & Internet Report: The Computer Software Journal for Scient*, 2(3):117–120, 2001.

[17] Qiagen. Ingenuity pathway analysis. http://biojs.io/d/biojs-vis-keggviewer. [Online; accessed 12-October-2015].

[18] Cytoscape Consortium. Cytoscape. `cytoscape.org`. [Online; accessed 05-June-2015].

[19] Alexander Lex, Marc Streit, Ernst Kruijff, and Dieter Schmalstieg. Caleydo: Design and evaluation of a visual analysis framework for gene expression data in its biological context. In *Pacific Visualization Symposium (PacificVis), 2010 IEEE*, pages 57–64. IEEE, 2010.

[20] The Caleydo Team. Caleydo. `caleydo.github.io`. [Online; accessed 05-November-2015].

[21] Alexander Lex, Christian Partl, Denis Kalkofen, Marc Streit, Samuel Gratzl, Anne Mai Wassermann, Dieter Schmalstieg, and Hanspeter Pfister. Entourage: Visualizing relationships between biological pathways using contextual subsets. *Visualization and Computer Graphics, IEEE Transactions on*, 19(12):2536–2545, 2013.

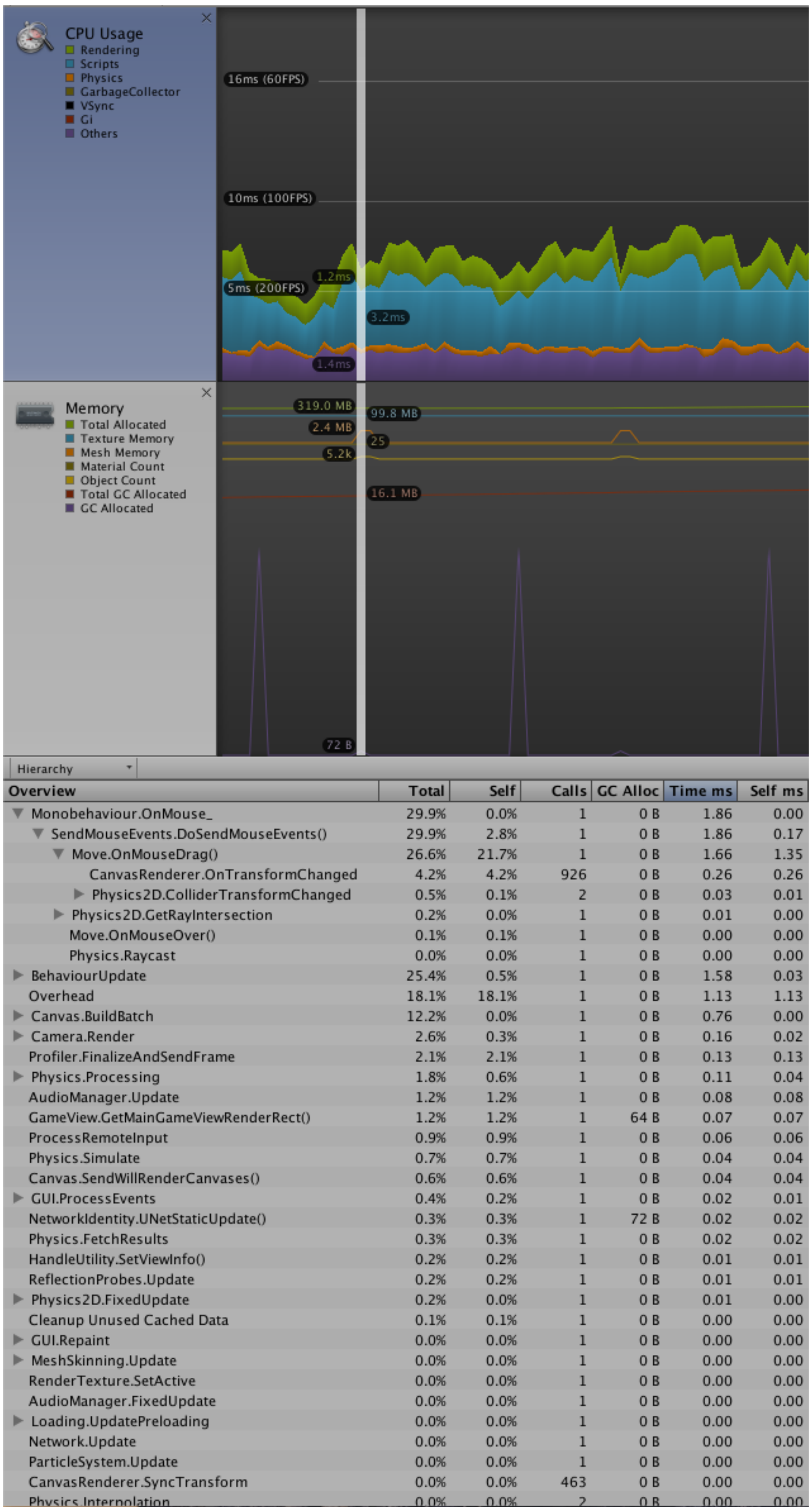
[22] Christian Partl, Alexander Lex, Marc Streit, Denis Kalkofen, Karl Kashofer, and Dieter Schmalstieg. enroute: dynamic path extraction from biological pathway maps for exploring heterogeneous experimental datasets. *BMC bioinformatics*, 14(Suppl 19):S3, 2013.

[23] Christopher Collins, Gerald Penn, and Sheelagh Carpendale. Bubble sets: Revealing set relations with isocontours over existing visualizations. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1009–1016, 2009.

[24] Christian Klukas and Falk Schreiber. Dynamic exploration and editing of kegg pathway diagrams. *Bioinformatics*, 23(3):344–350, 2007.

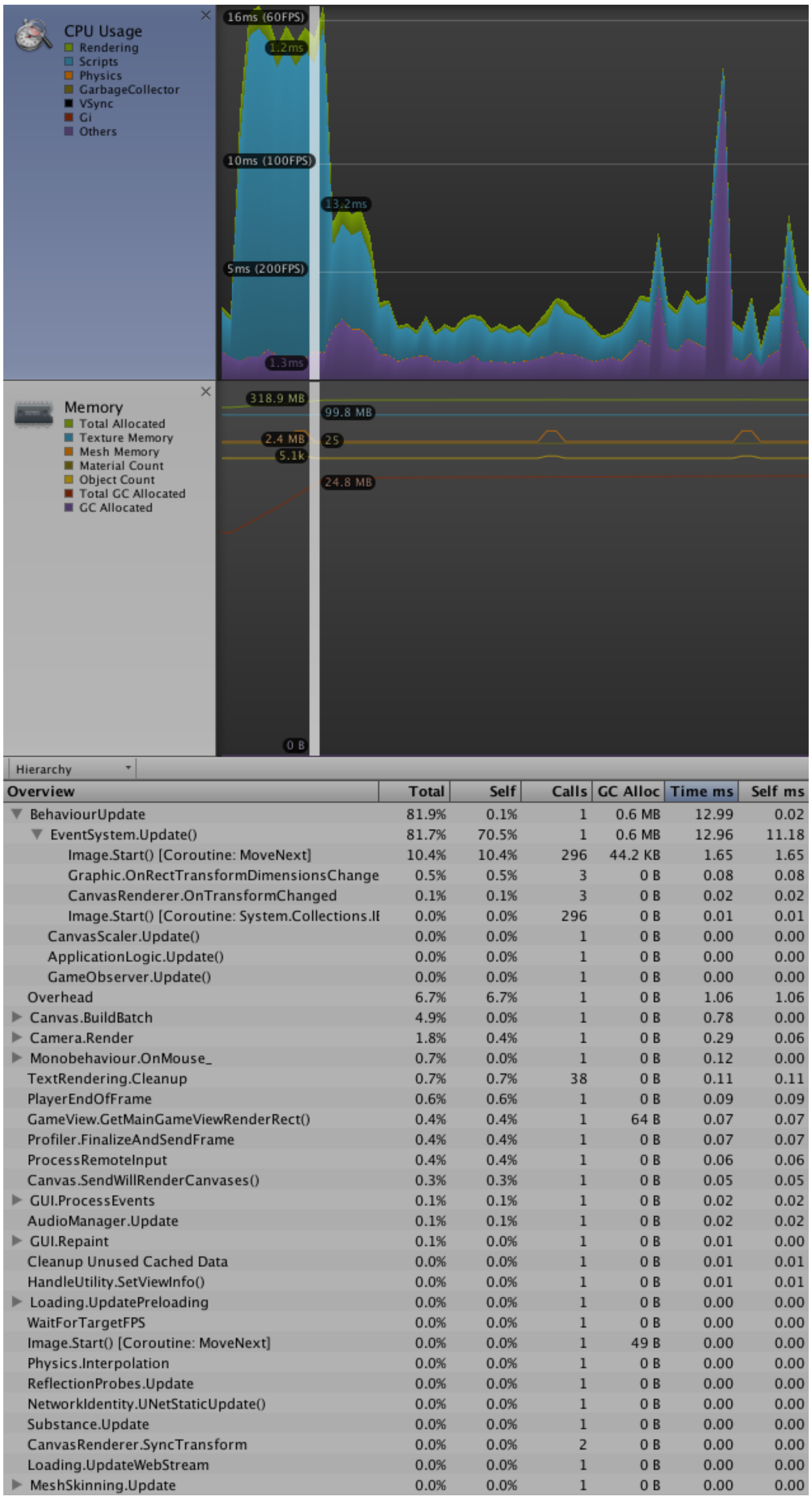
[25] Zhihan Lv, Alex Tek, Franck Da Silva, Charly Empereur-Mot, Matthieu Chavent, and Marc Baaden. Game on, science-how video game technology may help biologists tackle visualization challenges. *PloS one*, 8(3):57990, 2013.

Appendix A



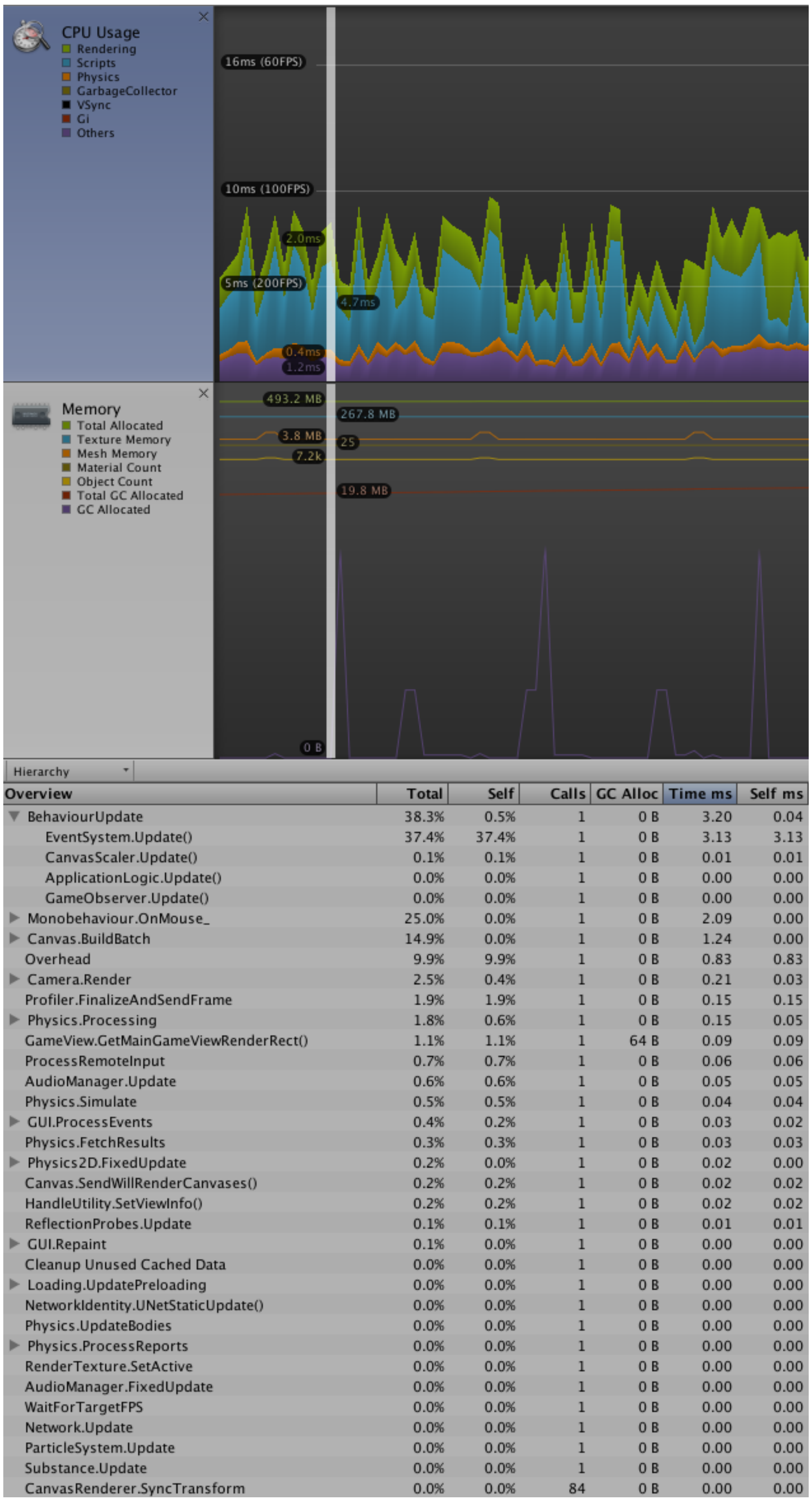
Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
▼ Monobehaviour.OnMouse_	29.9%	0.0%	1	0 B	1.86	0.00
▼ SendMouseEvents.DoSendMouseEvents()	29.9%	2.8%	1	0 B	1.86	0.17
▼ Move.OnMouseDrag()	26.6%	21.7%	1	0 B	1.66	1.35
CanvasRenderer.OnTransformChanged	4.2%	4.2%	926	0 B	0.26	0.26
▶ Physics2D.ColliderTransformChanged	0.5%	0.1%	2	0 B	0.03	0.01
▶ Physics2D.GetRayIntersection	0.2%	0.0%	1	0 B	0.01	0.00
Move.OnMouseOver()	0.1%	0.1%	1	0 B	0.00	0.00
Physics.Raycast	0.0%	0.0%	1	0 B	0.00	0.00
▶ BehaviourUpdate	25.4%	0.5%	1	0 B	1.58	0.03
Overhead	18.1%	18.1%	1	0 B	1.13	1.13
▶ Canvas.BuildBatch	12.2%	0.0%	1	0 B	0.76	0.00
▶ Camera.Render	2.6%	0.3%	1	0 B	0.16	0.02
Profiler.FinalizeAndSendFrame	2.1%	2.1%	1	0 B	0.13	0.13
▶ Physics.Processing	1.8%	0.6%	1	0 B	0.11	0.04
AudioManager.Update	1.2%	1.2%	1	0 B	0.08	0.08
GameView.GetMainGameViewRenderRect()	1.2%	1.2%	1	64 B	0.07	0.07
ProcessRemoteInput	0.9%	0.9%	1	0 B	0.06	0.06
Physics.Simulate	0.7%	0.7%	1	0 B	0.04	0.04
Canvas.SendWillRenderCanvases()	0.6%	0.6%	1	0 B	0.04	0.04
▶ GUI.ProcessEvents	0.4%	0.2%	1	0 B	0.02	0.01
NetworkIdentity.UNetStaticUpdate()	0.3%	0.3%	1	72 B	0.02	0.02
Physics.FetchResults	0.3%	0.3%	1	0 B	0.02	0.02
HandleUtility.SetViewInfo()	0.2%	0.2%	1	0 B	0.01	0.01
ReflectionProbes.Update	0.2%	0.2%	1	0 B	0.01	0.01
▶ Physics2D.FixedUpdate	0.2%	0.0%	1	0 B	0.01	0.00
Cleanup Unused Cached Data	0.1%	0.1%	1	0 B	0.00	0.00
▶ GUI.Repaint	0.0%	0.0%	1	0 B	0.00	0.00
▶ MeshSkinning.Update	0.0%	0.0%	1	0 B	0.00	0.00
RenderTexture.SetActive	0.0%	0.0%	1	0 B	0.00	0.00
AudioManager.FixedUpdate	0.0%	0.0%	1	0 B	0.00	0.00
▶ Loading.UpdatePreloading	0.0%	0.0%	1	0 B	0.00	0.00
Network.Update	0.0%	0.0%	1	0 B	0.00	0.00
ParticleSystem.Update	0.0%	0.0%	1	0 B	0.00	0.00
CanvasRenderer.SyncTransform	0.0%	0.0%	463	0 B	0.00	0.00
Physics.Interpolation	0.0%	0.0%	2	0 B	0.00	0.00

Figure 1: Screenshot of profiler with one pathways with drag operation.



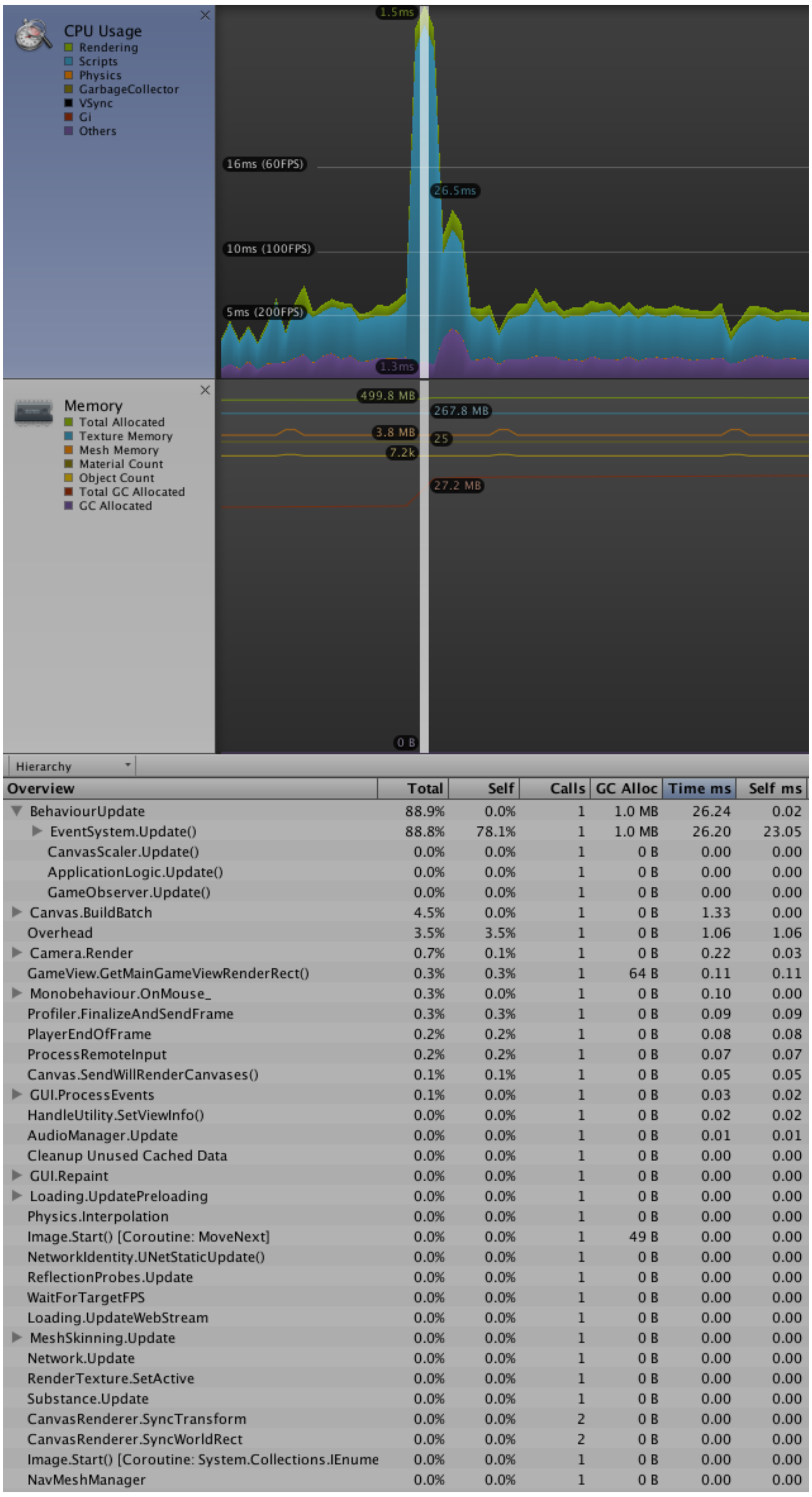
Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
▼ BehaviourUpdate	81.9%	0.1%	1	0.6 MB	12.99	0.02
▼ EventSystem.Update()	81.7%	70.5%	1	0.6 MB	12.96	11.18
Image.Start() [Coroutine: MoveNext]	10.4%	10.4%	296	44.2 KB	1.65	1.65
Graphic.OnRectTransformDimensionsChange	0.5%	0.5%	3	0 B	0.08	0.08
CanvasRenderer.OnTransformChanged	0.1%	0.1%	3	0 B	0.02	0.02
Image.Start() [Coroutine: System.Collections.II	0.0%	0.0%	296	0 B	0.01	0.01
CanvasScaler.Update()	0.0%	0.0%	1	0 B	0.00	0.00
ApplicationLogic.Update()	0.0%	0.0%	1	0 B	0.00	0.00
GameObserver.Update()	0.0%	0.0%	1	0 B	0.00	0.00
Overhead	6.7%	6.7%	1	0 B	1.06	1.06
▶ Canvas.BuildBatch	4.9%	0.0%	1	0 B	0.78	0.00
▶ Camera.Render	1.8%	0.4%	1	0 B	0.29	0.06
▶ Monobehaviour.OnMouse_	0.7%	0.0%	1	0 B	0.12	0.00
TextRendering.Cleanup	0.7%	0.7%	38	0 B	0.11	0.11
PlayerEndOfFrame	0.6%	0.6%	1	0 B	0.09	0.09
GameView.GetMainGameViewRenderRect()	0.4%	0.4%	1	64 B	0.07	0.07
Profiler.FinalizeAndSendFrame	0.4%	0.4%	1	0 B	0.07	0.07
ProcessRemoteInput	0.4%	0.4%	1	0 B	0.06	0.06
Canvas.SendWillRenderCanvases()	0.3%	0.3%	1	0 B	0.05	0.05
▶ GUI.ProcessEvents	0.1%	0.1%	1	0 B	0.02	0.02
AudioManager.Update	0.1%	0.1%	1	0 B	0.02	0.02
▶ GUI.Repaint	0.1%	0.0%	1	0 B	0.01	0.00
Cleanup Unused Cached Data	0.0%	0.0%	1	0 B	0.01	0.01
HandleUtility.SetViewInfo()	0.0%	0.0%	1	0 B	0.01	0.01
▶ Loading.UpdatePreloading	0.0%	0.0%	1	0 B	0.00	0.00
WaitForTargetFPS	0.0%	0.0%	1	0 B	0.00	0.00
Image.Start() [Coroutine: MoveNext]	0.0%	0.0%	1	49 B	0.00	0.00
Physics.Interpolation	0.0%	0.0%	1	0 B	0.00	0.00
ReflectionProbes.Update	0.0%	0.0%	1	0 B	0.00	0.00
NetworkIdentity.UNetStaticUpdate()	0.0%	0.0%	1	0 B	0.00	0.00
Substance.Update	0.0%	0.0%	1	0 B	0.00	0.00
CanvasRenderer.SyncTransform	0.0%	0.0%	2	0 B	0.00	0.00
Loading.UpdateWebStream	0.0%	0.0%	1	0 B	0.00	0.00
▶ MeshSkinning.Update	0.0%	0.0%	1	0 B	0.00	0.00

Figure 2: Screenshot of profiler with one pathways with gene expression operation.



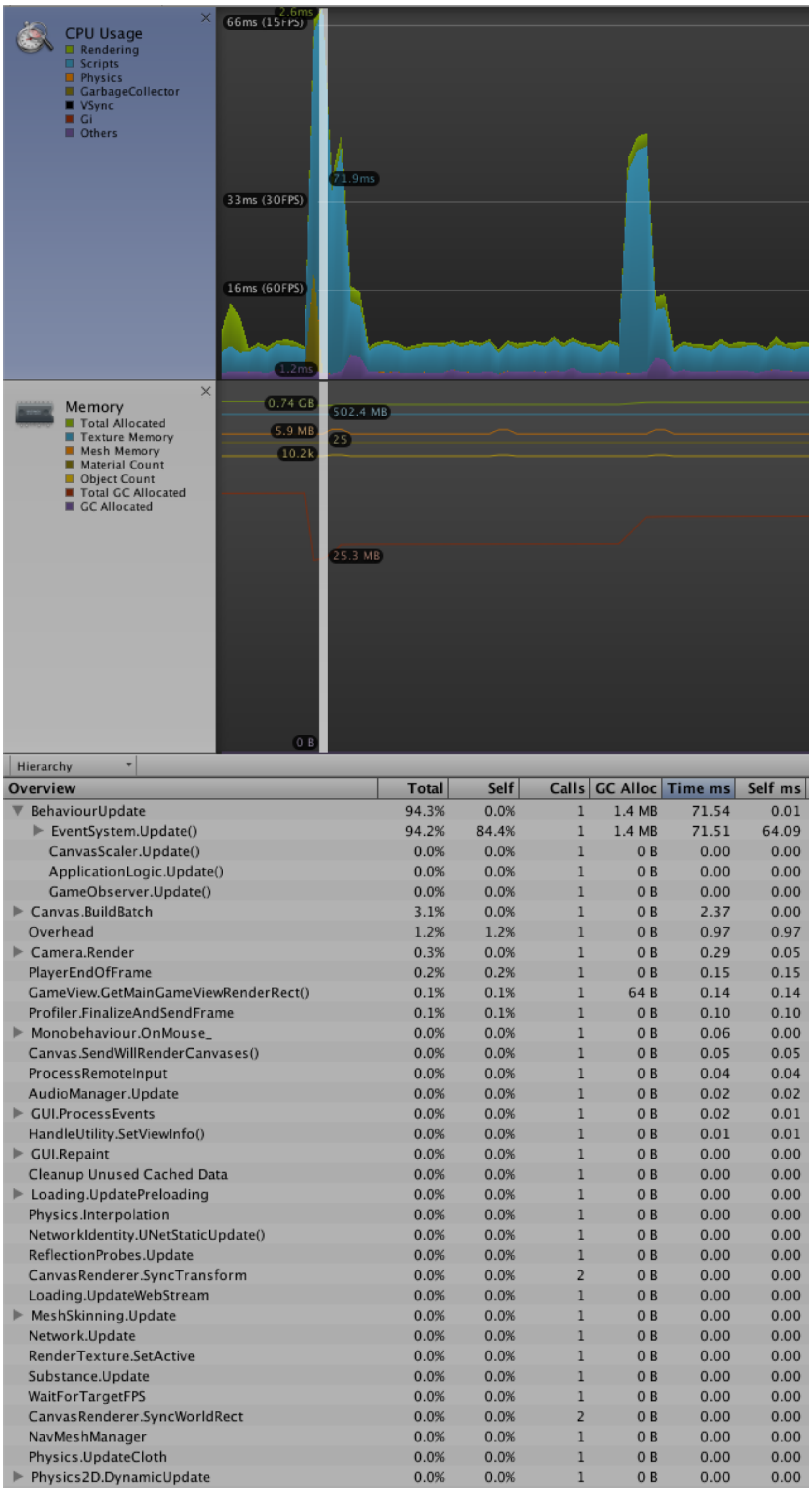
Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
▼ BehaviourUpdate	38.3%	0.5%	1	0 B	3.20	0.04
EventSystem.Update()	37.4%	37.4%	1	0 B	3.13	3.13
CanvasScaler.Update()	0.1%	0.1%	1	0 B	0.01	0.01
ApplicationLogic.Update()	0.0%	0.0%	1	0 B	0.00	0.00
GameObserver.Update()	0.0%	0.0%	1	0 B	0.00	0.00
▶ Monobehaviour.OnMouse_	25.0%	0.0%	1	0 B	2.09	0.00
▶ Canvas.BuildBatch	14.9%	0.0%	1	0 B	1.24	0.00
Overhead	9.9%	9.9%	1	0 B	0.83	0.83
▶ Camera.Render	2.5%	0.4%	1	0 B	0.21	0.03
Profiler.FinalizeAndSendFrame	1.9%	1.9%	1	0 B	0.15	0.15
▶ Physics.Processing	1.8%	0.6%	1	0 B	0.15	0.05
GameView.GetMainGameViewRenderRect()	1.1%	1.1%	1	64 B	0.09	0.09
ProcessRemoteInput	0.7%	0.7%	1	0 B	0.06	0.06
AudioManager.Update	0.6%	0.6%	1	0 B	0.05	0.05
Physics.Simulate	0.5%	0.5%	1	0 B	0.04	0.04
▶ GUI.ProcessEvents	0.4%	0.2%	1	0 B	0.03	0.02
Physics.FetchResults	0.3%	0.3%	1	0 B	0.03	0.03
▶ Physics2D.FixedUpdate	0.2%	0.0%	1	0 B	0.02	0.00
Canvas.SendWillRenderCanvases()	0.2%	0.2%	1	0 B	0.02	0.02
HandleUtility.SetViewInfo()	0.2%	0.2%	1	0 B	0.02	0.02
ReflectionProbes.Update	0.1%	0.1%	1	0 B	0.01	0.01
▶ GUI.Repaint	0.1%	0.0%	1	0 B	0.00	0.00
Cleanup Unused Cached Data	0.0%	0.0%	1	0 B	0.00	0.00
▶ Loading.UpdatePreloading	0.0%	0.0%	1	0 B	0.00	0.00
NetworkIdentity.UNetStaticUpdate()	0.0%	0.0%	1	0 B	0.00	0.00
Physics.UpdateBodies	0.0%	0.0%	1	0 B	0.00	0.00
▶ Physics.ProcessReports	0.0%	0.0%	1	0 B	0.00	0.00
RenderTexture.SetActive	0.0%	0.0%	1	0 B	0.00	0.00
AudioManager.FixedUpdate	0.0%	0.0%	1	0 B	0.00	0.00
WaitForTargetFPS	0.0%	0.0%	1	0 B	0.00	0.00
Network.Update	0.0%	0.0%	1	0 B	0.00	0.00
ParticleSystem.Update	0.0%	0.0%	1	0 B	0.00	0.00
Substance.Update	0.0%	0.0%	1	0 B	0.00	0.00
CanvasRenderer.SyncTransform	0.0%	0.0%	84	0 B	0.00	0.00

Figure 3: Screenshot of profiler with 7 pathways with drag operation.



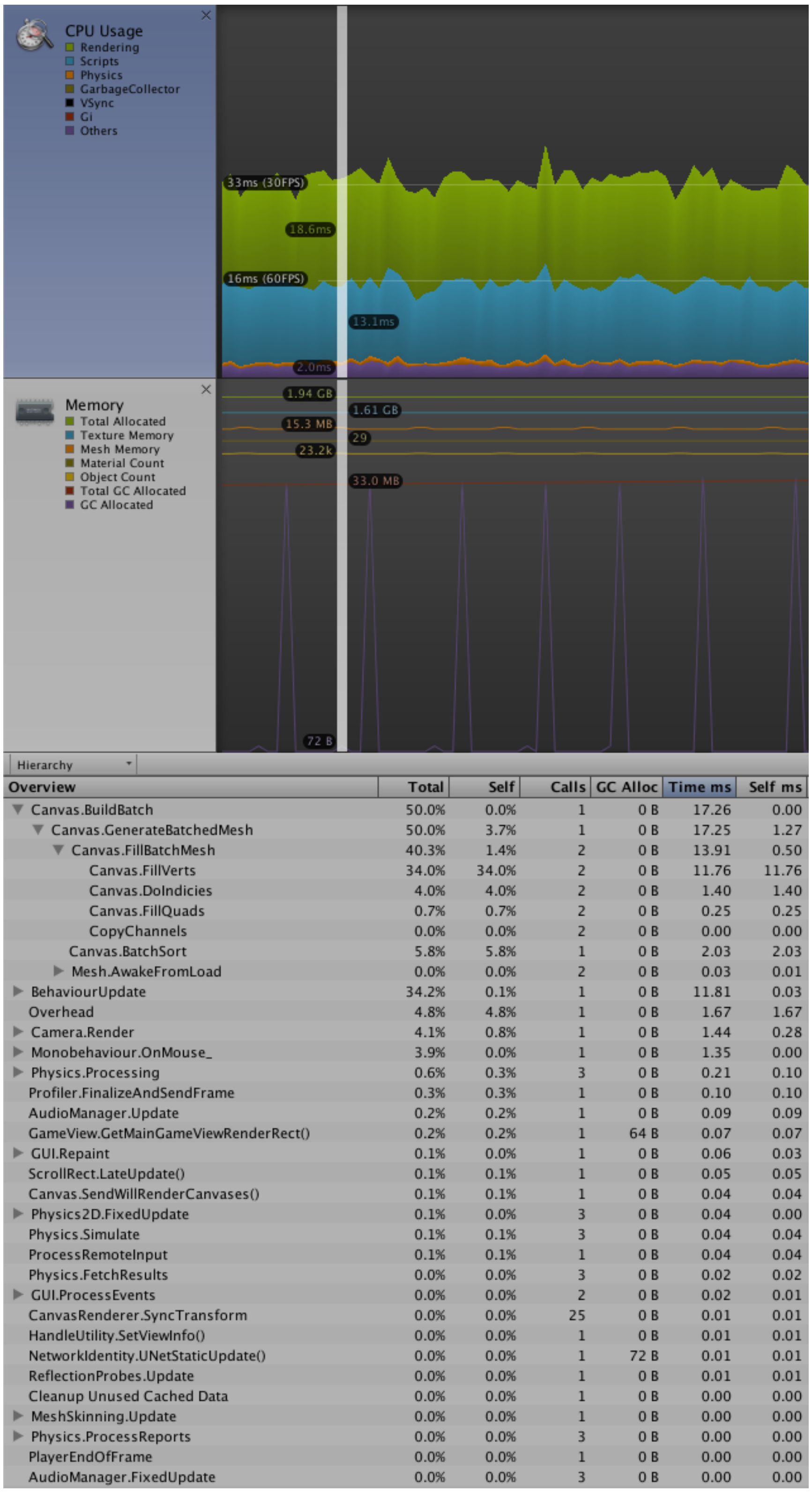
Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
BehaviourUpdate	88.9%	0.0%	1	1.0 MB	26.24	0.02
EventSystem.Update()	88.8%	78.1%	1	1.0 MB	26.20	23.05
CanvasScaler.Update()	0.0%	0.0%	1	0 B	0.00	0.00
ApplicationLogic.Update()	0.0%	0.0%	1	0 B	0.00	0.00
GameObserver.Update()	0.0%	0.0%	1	0 B	0.00	0.00
Canvas.BuildBatch	4.5%	0.0%	1	0 B	1.33	0.00
Overhead	3.5%	3.5%	1	0 B	1.06	1.06
Camera.Render	0.7%	0.1%	1	0 B	0.22	0.03
GameView.GetMainGameViewRenderRect()	0.3%	0.3%	1	64 B	0.11	0.11
Monobehaviour.OnMouse_	0.3%	0.0%	1	0 B	0.10	0.00
Profiler.FinalizeAndSendFrame	0.3%	0.3%	1	0 B	0.09	0.09
PlayerEndOfFrame	0.2%	0.2%	1	0 B	0.08	0.08
ProcessRemoteInput	0.2%	0.2%	1	0 B	0.07	0.07
Canvas.SendWillRenderCanvases()	0.1%	0.1%	1	0 B	0.05	0.05
GUI.ProcessEvents	0.1%	0.0%	1	0 B	0.03	0.02
HandleUtility.SetViewInfo()	0.0%	0.0%	1	0 B	0.02	0.02
AudioManager.Update	0.0%	0.0%	1	0 B	0.01	0.01
Cleanup Unused Cached Data	0.0%	0.0%	1	0 B	0.00	0.00
GUI.Repaint	0.0%	0.0%	1	0 B	0.00	0.00
Loading.UpdatePreloading	0.0%	0.0%	1	0 B	0.00	0.00
Physics.Interpolation	0.0%	0.0%	1	0 B	0.00	0.00
Image.Start() [Coroutine: MoveNext]	0.0%	0.0%	1	49 B	0.00	0.00
NetworkIdentity.UNetStaticUpdate()	0.0%	0.0%	1	0 B	0.00	0.00
ReflectionProbes.Update	0.0%	0.0%	1	0 B	0.00	0.00
WaitForTargetFPS	0.0%	0.0%	1	0 B	0.00	0.00
Loading.UpdateWebStream	0.0%	0.0%	1	0 B	0.00	0.00
MeshSkinning.Update	0.0%	0.0%	1	0 B	0.00	0.00
Network.Update	0.0%	0.0%	1	0 B	0.00	0.00
RenderTexture.SetActive	0.0%	0.0%	1	0 B	0.00	0.00
Substance.Update	0.0%	0.0%	1	0 B	0.00	0.00
CanvasRenderer.SyncTransform	0.0%	0.0%	2	0 B	0.00	0.00
CanvasRenderer.SyncWorldRect	0.0%	0.0%	2	0 B	0.00	0.00
Image.Start() [Coroutine: System.Collections.IEnume	0.0%	0.0%	1	0 B	0.00	0.00
NavMeshManager	0.0%	0.0%	1	0 B	0.00	0.00

Figure 4: Screenshot of profiler with 7 pathways with gene expression operation.



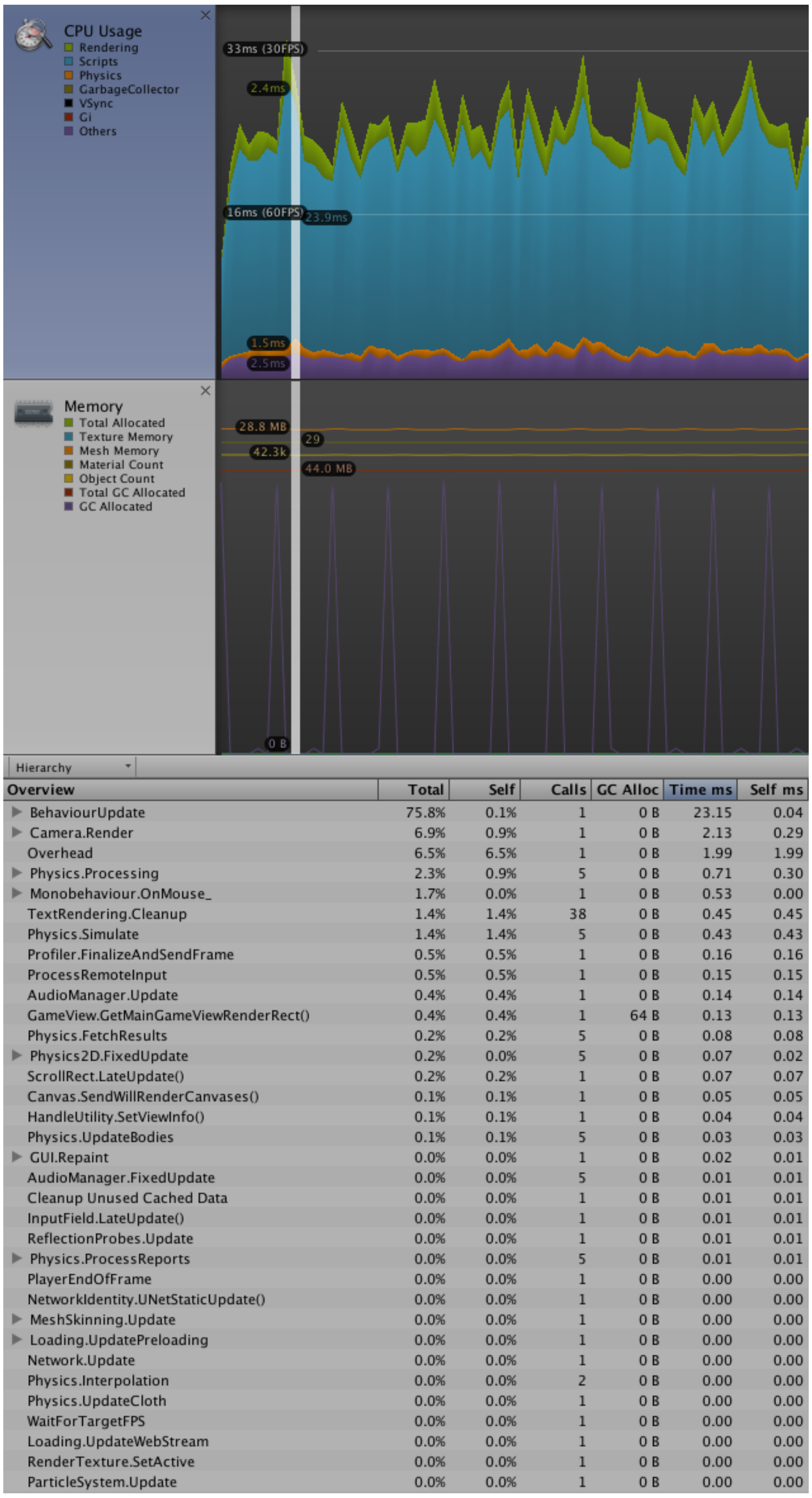
Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
▼ BehaviourUpdate	94.3%	0.0%	1	1.4 MB	71.54	0.01
▶ EventSystem.Update()	94.2%	84.4%	1	1.4 MB	71.51	64.09
CanvasScaler.Update()	0.0%	0.0%	1	0 B	0.00	0.00
ApplicationLogic.Update()	0.0%	0.0%	1	0 B	0.00	0.00
GameObserver.Update()	0.0%	0.0%	1	0 B	0.00	0.00
▶ Canvas.BuildBatch	3.1%	0.0%	1	0 B	2.37	0.00
Overhead	1.2%	1.2%	1	0 B	0.97	0.97
▶ Camera.Render	0.3%	0.0%	1	0 B	0.29	0.05
PlayerEndOfFrame	0.2%	0.2%	1	0 B	0.15	0.15
GameView.GetMainGameViewRenderRect()	0.1%	0.1%	1	64 B	0.14	0.14
Profiler.FinalizeAndSendFrame	0.1%	0.1%	1	0 B	0.10	0.10
▶ Monobehaviour.OnMouse_	0.0%	0.0%	1	0 B	0.06	0.00
Canvas.SendWillRenderCanvases()	0.0%	0.0%	1	0 B	0.05	0.05
ProcessRemoteInput	0.0%	0.0%	1	0 B	0.04	0.04
AudioManager.Update	0.0%	0.0%	1	0 B	0.02	0.02
▶ GUI.ProcessEvents	0.0%	0.0%	1	0 B	0.02	0.01
HandleUtility.SetViewInfo()	0.0%	0.0%	1	0 B	0.01	0.01
▶ GUI.Repaint	0.0%	0.0%	1	0 B	0.00	0.00
Cleanup Unused Cached Data	0.0%	0.0%	1	0 B	0.00	0.00
▶ Loading.UpdatePreloading	0.0%	0.0%	1	0 B	0.00	0.00
Physics.Interpolation	0.0%	0.0%	1	0 B	0.00	0.00
NetworkIdentity.UNetStaticUpdate()	0.0%	0.0%	1	0 B	0.00	0.00
ReflectionProbes.Update	0.0%	0.0%	1	0 B	0.00	0.00
CanvasRenderer.SyncTransform	0.0%	0.0%	2	0 B	0.00	0.00
Loading.UpdateWebStream	0.0%	0.0%	1	0 B	0.00	0.00
▶ MeshSkinning.Update	0.0%	0.0%	1	0 B	0.00	0.00
Network.Update	0.0%	0.0%	1	0 B	0.00	0.00
RenderTexture.SetActive	0.0%	0.0%	1	0 B	0.00	0.00
Substance.Update	0.0%	0.0%	1	0 B	0.00	0.00
WaitForTargetFPS	0.0%	0.0%	1	0 B	0.00	0.00
CanvasRenderer.SyncWorldRect	0.0%	0.0%	2	0 B	0.00	0.00
NavMeshManager	0.0%	0.0%	1	0 B	0.00	0.00
Physics.UpdateCloth	0.0%	0.0%	1	0 B	0.00	0.00
▶ Physics2D.DynamicUpdate	0.0%	0.0%	1	0 B	0.00	0.00

Figure 5: Screenshot of profiler with 14 pathways with drag operation.



Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
▼ Canvas.BuildBatch	50.0%	0.0%	1	0 B	17.26	0.00
▼ Canvas.GenerateBatchedMesh	50.0%	3.7%	1	0 B	17.25	1.27
▼ Canvas.FillBatchMesh	40.3%	1.4%	2	0 B	13.91	0.50
Canvas.FillVerts	34.0%	34.0%	2	0 B	11.76	11.76
Canvas.DoIndicies	4.0%	4.0%	2	0 B	1.40	1.40
Canvas.FillQuads	0.7%	0.7%	2	0 B	0.25	0.25
CopyChannels	0.0%	0.0%	2	0 B	0.00	0.00
Canvas.BatchSort	5.8%	5.8%	1	0 B	2.03	2.03
▶ Mesh.AwakeFromLoad	0.0%	0.0%	2	0 B	0.03	0.01
▶ BehaviourUpdate	34.2%	0.1%	1	0 B	11.81	0.03
Overhead	4.8%	4.8%	1	0 B	1.67	1.67
▶ Camera.Render	4.1%	0.8%	1	0 B	1.44	0.28
▶ Monobehaviour.OnMouse_	3.9%	0.0%	1	0 B	1.35	0.00
▶ Physics.Processing	0.6%	0.3%	3	0 B	0.21	0.10
Profiler.FinalizeAndSendFrame	0.3%	0.3%	1	0 B	0.10	0.10
AudioManager.Update	0.2%	0.2%	1	0 B	0.09	0.09
GameView.GetMainGameViewRenderRect()	0.2%	0.2%	1	64 B	0.07	0.07
▶ GUI.Repaint	0.1%	0.0%	1	0 B	0.06	0.03
ScrollRect.LateUpdate()	0.1%	0.1%	1	0 B	0.05	0.05
Canvas.SendWillRenderCanvases()	0.1%	0.1%	1	0 B	0.04	0.04
▶ Physics2D.FixedUpdate	0.1%	0.0%	3	0 B	0.04	0.00
Physics.Simulate	0.1%	0.1%	3	0 B	0.04	0.04
ProcessRemoteInput	0.1%	0.1%	1	0 B	0.04	0.04
Physics.FetchResults	0.0%	0.0%	3	0 B	0.02	0.02
▶ GUI.ProcessEvents	0.0%	0.0%	2	0 B	0.02	0.01
CanvasRenderer.SyncTransform	0.0%	0.0%	25	0 B	0.01	0.01
HandleUtility.SetViewInfo()	0.0%	0.0%	1	0 B	0.01	0.01
NetworkIdentity.UNetStaticUpdate()	0.0%	0.0%	1	72 B	0.01	0.01
ReflectionProbes.Update	0.0%	0.0%	1	0 B	0.01	0.01
Cleanup Unused Cached Data	0.0%	0.0%	1	0 B	0.00	0.00
▶ MeshSkinning.Update	0.0%	0.0%	1	0 B	0.00	0.00
▶ Physics.ProcessReports	0.0%	0.0%	3	0 B	0.00	0.00
PlayerEndOfFrame	0.0%	0.0%	1	0 B	0.00	0.00
AudioManager.FixedUpdate	0.0%	0.0%	3	0 B	0.00	0.00

Figure 6: Screenshot of profiler with 50 pathways with drag operation.



Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
▶ BehaviourUpdate	75.8%	0.1%	1	0 B	23.15	0.04
▶ Camera.Render	6.9%	0.9%	1	0 B	2.13	0.29
Overhead	6.5%	6.5%	1	0 B	1.99	1.99
▶ Physics.Processing	2.3%	0.9%	5	0 B	0.71	0.30
▶ Monobehaviour.OnMouse_	1.7%	0.0%	1	0 B	0.53	0.00
TextRendering.Cleanup	1.4%	1.4%	38	0 B	0.45	0.45
Physics.Simulate	1.4%	1.4%	5	0 B	0.43	0.43
Profiler.FinalizeAndSendFrame	0.5%	0.5%	1	0 B	0.16	0.16
ProcessRemoteInput	0.5%	0.5%	1	0 B	0.15	0.15
AudioManager.Update	0.4%	0.4%	1	0 B	0.14	0.14
GameView.GetMainGameViewRenderRect()	0.4%	0.4%	1	64 B	0.13	0.13
Physics.FetchResults	0.2%	0.2%	5	0 B	0.08	0.08
▶ Physics2D.FixedUpdate	0.2%	0.0%	5	0 B	0.07	0.02
ScrollRect.LateUpdate()	0.2%	0.2%	1	0 B	0.07	0.07
Canvas.SendWillRenderCanvases()	0.1%	0.1%	1	0 B	0.05	0.05
HandleUtility.SetViewInfo()	0.1%	0.1%	1	0 B	0.04	0.04
Physics.UpdateBodies	0.1%	0.1%	5	0 B	0.03	0.03
▶ GUI.Repaint	0.0%	0.0%	1	0 B	0.02	0.01
AudioManager.FixedUpdate	0.0%	0.0%	5	0 B	0.01	0.01
Cleanup Unused Cached Data	0.0%	0.0%	1	0 B	0.01	0.01
InputField.LateUpdate()	0.0%	0.0%	1	0 B	0.01	0.01
ReflectionProbes.Update	0.0%	0.0%	1	0 B	0.01	0.01
▶ Physics.ProcessReports	0.0%	0.0%	5	0 B	0.01	0.01
PlayerEndOfFrame	0.0%	0.0%	1	0 B	0.00	0.00
NetworkIdentity.UNetStaticUpdate()	0.0%	0.0%	1	0 B	0.00	0.00
▶ MeshSkinning.Update	0.0%	0.0%	1	0 B	0.00	0.00
▶ Loading.UpdatePreloading	0.0%	0.0%	1	0 B	0.00	0.00
Network.Update	0.0%	0.0%	1	0 B	0.00	0.00
Physics.Interpolation	0.0%	0.0%	2	0 B	0.00	0.00
Physics.UpdateCloth	0.0%	0.0%	1	0 B	0.00	0.00
WaitForTargetFPS	0.0%	0.0%	1	0 B	0.00	0.00
Loading.UpdateWebStream	0.0%	0.0%	1	0 B	0.00	0.00
RenderTexture.SetActive	0.0%	0.0%	1	0 B	0.00	0.00
ParticleSystem.Update	0.0%	0.0%	1	0 B	0.00	0.00

Figure 7: Screenshot of profiler with 100 pathways running.