Элементы систем научной визуализации

П.А. Васёв[1], М.О. Бахтерев[1], Д.В. Манаков[1], С.В. Поршнев [1,2]

[1] *ИММ УрО РАН им. Н.Н. Красовского, ул. Софьи Ковалевской 16, Екатеринбург, 620108, Россия*
[2] *Уральский федеральный университет им. Б.Н. Ельцина», ул. Мира 19, Екатеринбург, 620002, Россия*

Аннотация

В работе представлен обзор некоторых элементов логических моделей пользовательских интерфейсов систем научной визуализации. При разработке как универсальных, так и специализированных систем визуализации возникает вопрос, какие методы организации работы пользователя можно положить в основу интерфейсов этих систем. Цель – предложить пользователю набор инструментов, позволяющих гибко формировать визуальное представление изучаемых явлений. Некоторые из этих методов — общеприменимы, другие – специфичны для задач визуализации. В работе описана идея модификаторов, которая часто применяется в программировании для формирования поведения и внешнего вида цифровых объектов. Рассмотрена идея расширений, применяемых для подключения внешних модулей к системам, в том числе создаваемых сторонними разработчиками. Рассмотрены вопросы программирования динамики в сценах визуализации, в том числе, с учётом опыта игровых средств разработки. Приведены модели использования данных в проектах визуализации, передачи их в модули визуализации, организации дополнительных вычислений в интересах визуализации. Также в работе представлена идея оценки визуальных языков с точки зрения их выразительности, аналогично подходу, который применяется к обычным языкам программирования.

Ключевые слова [1]

Научная визуализация, система визуализации, логическая модель, интерфейс пользователя.

Elements of Scientific Visualization Systems

Pavel Vasev[1], Mikhail Bakhterev[1], Dmitry Manakov[1], Sergey Porshnev [1,2]

[1] *N.N. Krasovskii Institute of Mathematics and Mechanics of the Ural Branch of the Russian Academy of Sciences, 16 S. Kovalevskaya Str., Ekaterinburg, 620108, Russia*
[2] *Ural Federal University named after B.N. Yeltsin, 19 Mira street, Ekaterinburg, 620002, Russia*

Abstract

The paper discusses elements of logical models of graphical user interfaces used in both universal and specialized scientific visualization systems. Criteria of expressiveness of programming language that are discussed in “Structure and Interpretation of Computer Programs” book are applied to graphical interfaces. It is shown that graphical interfaces allow user to operate on same digital substance and with same logical approaches as in textual programming languages. Both use basic elements, allow their combination, and support the procedure of abstraction. Authors suggest considering this aspect when developing graphical interfaces. Then, idea of modifiers (known also as behaviors, effects, so on) is discussed. Idea of extensions (known also as plugins, modules, and applications) is also discussed. Some methods of programming of scene dynamics are presented. Also languages and ontologies of scientific visualization are discussed, e.g. models for editing visualization pipeline: adding data to projects, filtering of that data, and methods of description of data representation on screen.

ГрафиКон 2022: 32-я Международная конференция по компьютерной графике и машинному зрению, 19-22 сентября 2022 г., Рязанский государственный радиотехнический университет им. В.Ф. Уткина, Рязань, Россия
EMAIL: vasev@imm.uran.ru (П.А. Васёв); m.bakhterev@imm.uran.ru (М.О. Бахтерев); manakov@imm.uran.ru (Д.В. Манаков); s.v.porshnev@urfu.ru (С.В. Поршнев)
ORCID: 0000-0003-3854-0670 (П.А. Васёв); 0000-0001-8016-9946 (М.О. Бахтерев); 0000-0001-6852-8096 (Д.В. Манаков); 0000-0001-6884-9033 (С.В. Поршнев)
© 2022 Copyright for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Keywords
Scientific visualization, visualization system, logical model, user interface.

1. Введение

Для решения задач научной визуализации применяются два вида систем – универсальные системы визуализации (системы общего назначения) и специализированные системы. Универсальные системы позволяют собрать (запрограммировать) конвейер визуализации из готовых блоков. У этого подхода есть свои преимущества. Однако универсальные системы обладают следующими принципиальными недостатками.

Во-первых, при решении конкретной задачи визуализации работа пользователя может потребовать избыточных трудозатрат на работу, которая могла бы быть автоматизирована. Например, можно было бы по открываемому файлу автоматически определять тип его содержимого и рисовать некоторый подходящий визуальный образ. Но обычно выбор визуального образа требует длительного прохода по нетривиальному набору диалоговых окон.

Во-вторых, универсальная система визуализации может не содержать необходимых методов отображения информации. Доступные методы могут скрывать особенности данных, важные в конкретной задаче визуализации.

Для решения этих проблем создаются специализированные системы визуализации. Они повышают эффективность пользователя в работе над его конкретной задачей визуализации и позволяют акцентировать внимание именно на тех особенностях, которые важны для данной задачи, обеспечивая корректность и информативность визуализации [1,2].

Отметим, что реальные системы визуализации обладают чертами как универсальных, так и специализированных систем. Это связано с тем, что даже специализированные системы, созданные под решение конкретных задач, по мере применения нуждаются в дальнейшем развитии, доработке под новые запросы пользователей. Обычно подобное «допрограммирование» приходится выполнять специалисту – программисту средств визуализации.

Однако существенную часть «допрограммирования» можно передать в руки пользователей, если предоставить им подходящий инструментарий. Поэтому в специализированные системы вносят элементы универсальных систем визуализации. Пользователи в этом случае получают возможность самостоятельно конструировать алгоритм визуализации, строить больше разнообразных визуальных образов, проверять больше гипотез. Потенциально, это позволяет быстрее и качественнее достигать цели визуализации — понимания исследуемого явления.

Возникают вопросы о том, какие именно элементы, логические модели, подходы можно выделить в существующих универсальных системах и эффективно применить при создании новых систем визуализации. В настоящей работе мы представляем свои ответы на эти вопросы.

2. Варианты программирования визуализации

Программировать визуализацию (а точнее, виды отображения, определение дано в [3]) можно в текстовых языках программирования (ЯП) или с помощью визуальных средств. В рамках обоих подходов существуют как универсальные, так и специализированные под задачи визуализации инструменты. Рассмотрим возможные варианты подробнее.

Программирование в универсальных текстовых языках программирования:

К1. Универсальные коды. Программу разрабатывают как отдельный проект под конкретную задачу визуализации. Зачастую так решают небольшие задачи визуализации. Разработку на ЯП общего назначения облегчают библиотеки графики и визуализации, например, для языка Python – это Matplotlib, Kitware trame, Vedo.embl.es, для языка JavaScript – Three.js, d3.js, Babylon.js.

К2. Универсальные коды, использующие программные каркасы (frameworks). Каркасы отличаются от библиотек тем, что в большой степени определяют поток управления в программе, реализуя принцип инверсии управления (inversion of control). Авторы создали несколько подобных каркасов: Sharpeye [18], JUJ [19] и др. Также к этой категории относятся популярные каркасы A-frame, Unity, Unreal engine и другие.

К3. Специализированные коды. Программу визуализации записывают на предметно-ориентированных языках (DSL) визуализации таких, как Vega grammar и VRML.

Варианты **К1** и **К2** подходят для специалистов по программированию. Вариант **К3** тоже ориентирован преимущественно на них – как средство автоматизации труда. Но вариант **К3** может быть удобен и для пользователей визуализации.

Визуальное программирование:

В1. Графовые среды визуального программирования. Работая в этом классе систем, пользователь добавляет на плоскость символы различных сущностей, налаживает связи между ними, проводит настройку параметров сущностей. Система сущностей и связей между ними интерпретируются в некоторой модели вычислений, например, в модели потока данных (data flow). В результате формируется процесс, реализующий конвейер требуемой визуализации. Примеры подобных сред: DataExplorer, Enso, XcluDev, SciVi [13].

В2. Визуальные среды с жёсткой моделью. Например, известные программы Paraview, Visit, Blender. В этих средах пользователь зачастую лишён возможности добавлять произвольные коды в конвейер визуализации, но может настраивать (программировать) визуализацию, используя команды визуального интерфейса (кнопки, меню) или подсистему сценариев в рамках предлагаемой средой модели программирования.

Вариант **В1** подходит и пользователям, и специалистам по визуализации (хотя, кажется, специалисты в программировании чаще предпочитают текстовое представление программ). Вариант **В2** подходит больше пользователям визуализации, чем специалистам.

Проиллюстрируем дополнительно тезис о доработке систем визуализации, высказанный во введении. Привлечение к разработке визуализации специалиста-программиста имеет следующую особенность. Обычно результат работы такого программиста – это программа (специализированная система визуализации), которая решает поставленную задачу, но которая сама не является инструментом программирования. Пользователь может загрузить данные, программа покажет заранее сконструированные виды отображения, предоставит возможности их настроек. Но, как правило, возможности такой программы этим и ограничены, дальше настроек дело не идёт.

Например, пользователь не может добавить новый экземпляр некоторой визуальной сущности, скажем, второй системы координат, потому что программа предусматривает отображение только одного экземпляра системы координат. Если пользователю потребуется добавить количественно или качественно новые визуальные образы, ему придётся обращаться с этим вопросом к специалисту, потому что необходимые для этого механизмы не включены в программу.

Привлечение специалистов для разработки визуализации – распространённая практика. Её преимущество — продолжение междисциплинарного диалога, взаимодействия и обмена информацией. Её недостатки — зависимость пользователя от специалиста, повышенная стоимость разработки, задержки во времени.

Эти недостатки можно нивелировать, предоставив пользователю инструменты для самостоятельного быстрого решения новых задач визуализации, возникающих в процессе исследования.

Итак, наш основной интерес в представленной работе – это пользовательское программирование научной визуализации. Наш опыт работы с различными вариантами такого программирования подсказывает сфокусировать внимание на методах визуального программирования, обозначенных в **В2.** Сообществом за множество лет развития программных систем компьютерной графики выработан ряд методов такого программирования. Далее в работе представлен анализ некоторых из них.

3. Выразительность визуальных языков

Взглянем на работу пользователя со следующего ракурса. Во многих случаях пользователь, работая с той или иной программой, формирует *цифровой артефакт*. Цифровыми артефактами мы собирательно называем информационные сущности в цифровой форме: текст, видео, презентацию, базу данных и т.п. Типичная программа с графическим интерфейсом,

предназначенная для формирования цифровых артефактов, предоставляет пользователю визуальный язык: сам графический интерфейс и его семантику — логическую модель интерпретации действий пользователя. Так, работая с системой визуализации, пользователь формирует необходимый ему набор визуальных образов, конструируя вид отображения.

Конструкция: интерфейс (представление) и его семантика – похожа на структуру традиционных языков программирования: исходный текст (представление) и его семантика. Для последних сформулированы необходимые условия выразительности, см. [4, 5].

Выразительный язык программирования должен обладать следующими свойствами.

S1. Предоставлять набор элементарных базовых сущностей.

S2. Позволять конструировать более сложные сущности комбинированием более простых.

S3. Обеспечивать абстрагирование и именование полученных комбинаций, чтобы затем их можно было использовать как единое целое в новых комбинациях.

Чем полнее ЯП реализует возможности перечня **S1-3**, тем он выразительнее. Мы можем спроецировать эти свойства на работу пользователя с программами с графическими интерфейсами. Это позволит понять, какими возможностями следует наделить визуальный язык (интерфейс) программы. Чем полнее эти возможности соответствуют **S1-3**, тем выразительнее будет программа, тем больше результатов пользователь сможет в ней получить.

Например, рассмотрим работу с Powerpoint. Пользователь может вводить разные виды базовых элементарных визуальных объектов: тексты, картинки, геометрические фигуры, анимации. Это свойство **S1**. Эти объекты могут быть скомбинированы в слайды и могут быть объединены в группы. Это свойство **S2**. Свойство **S3**, абстрагирование, – реализовано не в полной мере, но, всё же, присутствует. Можно выделять и копировать группы объектов. Можно считать, что для пользователя это является операцией абстрагирования: о выбранных объектах можно мыслить, как о едином целом. Отметим, что изменения исходной конструкции не будут отражены в её копиях. Мы считаем, это означает, что полноценно на этом уровне свойство абстракции в Powerpoint не реализовано: процессы взаимодействия пользователей невозможно абстрагировать.

На уровне презентаций абстрагирование реализовано в более полной мере. Например, пользователь может создать шаблон дизайна презентации и использовать его в различных проектах. Изменение шаблона повлечёт изменение использующих его презентаций. Можно считать, что в Powerpoint реализовано несколько визуальных языков для работы на разных уровнях: работа с визуальными объектами, работа на уровне слайдов, работа на уровне презентаций, работа на уровне шаблонов. И в этих языках свойства **S1-3** реализованы в разной степени.

Проводя параллели между текстовыми ЯП и визуальными языками, к которым мы относим и графические интерфейсы, можно сделать вывод о том, что пользователь, конструируя цифровые артефакты, занимается программированием. Цифровые артефакты состоят из сущностей, схожих с сущностями, которые описывают коды на текстовых ЯП: базовые элементы и их комбинации, абстракции и их применения. Мы можем подкрепить этот вывод тем, что одинаковый результат часто можно получить обоими способами — через программирование как на текстовом языке, так и через работу на визуальном.

Свойства **S1-3** можно спроецировать и на графические интерфейсы программ научной визуализации. В простых системах визуализации, выполняющих визуализацию в рамках жёстко заданного вида отображения, мы их не обнаружим. Более «продвинутые» системы – реализуют некоторые элементы **S1-3**. Наиболее полно проявляют эти свойства универсальные системы визуализации.

4. Модификаторы

Обычно семантика развитого текстового ЯП определяет разные методы комбинирования сущностей: формирование структур данных из отдельных значений, передача значений при обращении к подпрограммам, сборка функций в материализации (reify) интерфейсов, комбинации классов и интерфейсов при создании производных классов, помещение вычислений в ленивые (lazy) или асинхронные (await) контексты и т.п. Так и в визуальных языках (в

частности в языках графических интерфейсов) можно обнаружить различные способы комбинации сущностей. Один из таких способов — механизм *модификаторов*. Другие названия этого понятия – компоненты (components), эффекты (effects), поведения (behaviours), стили.

Модификатор – дополнительная сущность, «прикрепляемая» к основной сущности. Модификатор может влиять на поведение основной сущности. Такое влияние может быть как косметическим, так и существенным. Примеры модификаторов.

1. Модификатор «следовать за объектом», параметризованный *целевым объектом,* управляет положением своего основного объекта так, чтобы по мере перемещения целевого основной оказывался бы рядом с ним. Возможны разные способы вычисления позиции основного объекта: он может жёстко выдерживать расстояние до целевого, или расстояние может быть эластичным. Такой модификатор можно прикреплять к разным объектам, например, к объекту камеры.
2. Модификатор «сечение» модифицирует изображение основного визуального объекта так, чтобы показывать только его часть, например, находящуюся в заданном полупространстве. Модификатор может добавить в сцену новый визуальный объект, например, в данном случае — плоскость, задающую сечение. Эти объекты могут упрощать настройку параметров модификатора через *прямое управление* (direct manipulation).
3. Модификатор «поверхность земли» («terrain» в среде разработки Unity [6], см. рисунок 1) прикрепляется к объекту некоторой плоскости в сцене и предоставляет интерфейс для формирования из этой плоскости холмистой поверхности (её формы и текстуры) и для размещения на ней экземпляров моделей растений. Используется прямое управление – пользователь формирует холмы и размещает растения, работая с визуальным образом поверхности прямо в основном окне редактора Unity.

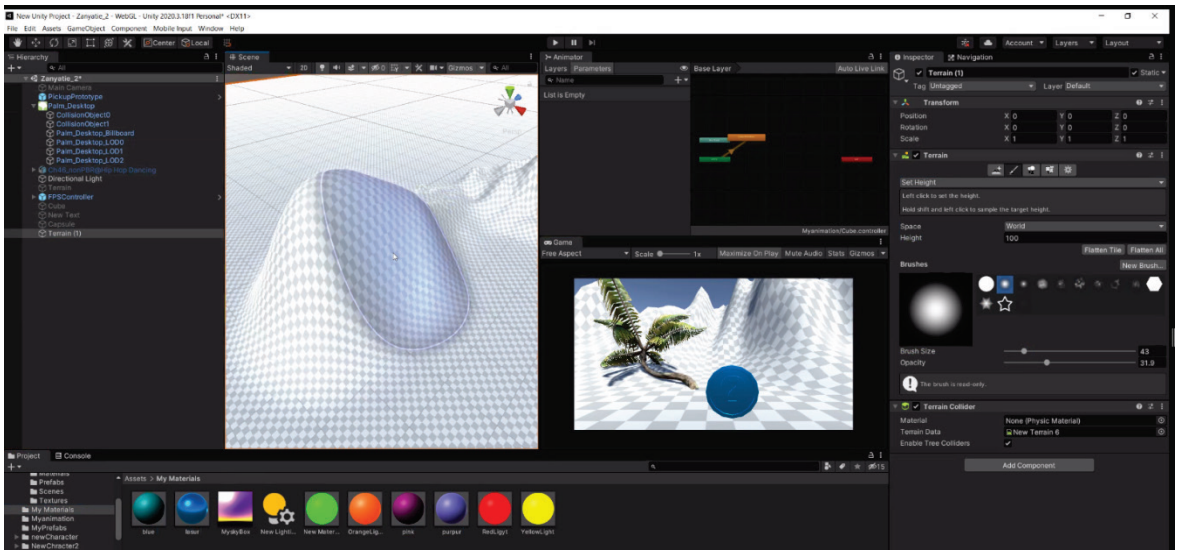

Рисунок 1 – Применение модификатора “Terrain” в среде Unity. Графический интерфейс модификатора находится справа, вкладка Terrain. Рядом видны интерфейсы других модификаторов, Transform и Terrain Collider. Внизу справа кнопка Add Component – для добавления новых модификаторов. Автор снимка – А. Е. Крохалева

Пример «terrain» показывает, что модификатор, не смотря на свою подчинённую роль по отношению к основному объекту, может обладать самостоятельным сложным интерфейсом и богатыми возможностями.

Технически, модификатор — это полноценная программная компонента, которая взаимодействует с основным объектом и с другими прикреплёнными к нему модификаторами, через определённые программные интерфейсы.

Аналогом модификаторов в текстовых ЯП можно считать миксины (mixin) в парадигме объектно-ориентированного программирования, оснащённые параметрами и применяемые не

только к классам, но и к экземплярам объектов. Другим аналогом являются модификаторы (modifiers) в каркасах Compose Multiplatform (JetBrains) и SwiftUI (Apple). Программирование с модификаторами — это одна из реализаций принципа «композиция вместо наследования» (composition over inheritance).

В предельном варианте программирования с модификаторами роль основных объектов сужается до контейнеров модификаторов, без иной собственной функциональности. Например, так устроена модель Entity-Component-System [7,8], в которой объекты пусты, модификаторы служат метками и носителями значений, а алгоритмический слой реализован в системах. При этом, пользователь формирует набор активных систем отдельно.

Модификатор для реализации своего поведения может создавать другие модификаторы и привязывать их к объектам. Например, на такого рода модификаторы опирается проект A-Frame, в котором они названы *higher-order components* (аналог higher-order functions), см. [9].

Кроме этого, мы считаем, что модификаторы являются способом реализации единообразности элементов визуального языка интерфейса пользователя. Представленный в интерфейсе одинаково для разных объектов модификатор, что указывает на одинаковый его «смысл» для пользователя, может быть запрограммирован по-своему для каждого из типов объектов.

Во многих системах визуального программирования есть интересная возможность применения модификаторов не только к отдельным объектам, но и к множествам объектов. Это может быть реализовано, например, такой моделью работы пользователя: сначала объект из библиотеки добавляют на панель проекта, на этой панели к нему прикрепляют модификаторы и меняют его свойства, после чего уже в сцене создают конкретные экземпляры полученной на панели проекта конструкции. Таким образом системы позволяют визуально формировать «прокси-классы» и создавать их экземпляры.

Иногда в интерфейсах визуальные модификаторы разделяются на категории. Например, в среде Gdevelop это категории «поведение» и «визуальные эффекты».

Мы считаем, что применение модификаторов повышает выразительные возможности языков (не только визуальных), предназначенных для формирования цифровых артефактов. Они упрощают программирование конкретных элементов артефакта, поскольку обобщённые элементарные поведения (модификации) объектов могут быть вынесены в отдельный слой кода или визуального рабочего пространства, и могут быть применены к различным типам объектов, средствами графического или текстового интерфейса пользователя.

5. Программирование динамики

Интерфейс конструирования сложных цифровых артефактов, которыми являются сцены визуализации, должен охватывать семантически разные направления. В каждом из этих направлений должны быть реализованы возможности **S1-3** в необходимом объёме.

Рассмотрим в этой части работы проблемы программирования динамики сцены визуализации, то есть, описания процессов изменения состояний сцены: например, анимации развития во времени изображаемого физического явления.

Мы предполагали, что достаточно реализовывать компоненты динамики сцены в добавляемых в неё объектах или в их модификаторах. Такой подход используют многие игровые системы. Они предоставляют пользователю поведенческие модификаторы (behaviours) из предопределённого набора. Примеры поведенческих модификаторов: уже упоминавшийся модификатор «следуй за объектом» или «объект модели столкновений» (collider) и т.п. Прикрепляя такие модификаторы к объекту, можно добавлять к объектам заранее определённые виды поведений. Однако оказалось, что на практике этот подход недостаточно удобен, чтобы описывать все необходимые виды динамики в сценах.

Существуют и другие подходы. Динамическое поведение можно задавать функциональной зависимостью *свойств* (параметров, переменных) визуальных объектов от «времени». Например, в Unity эту зависимость задают для выбранного объекта указанием ключевых точек на оси времени и значений свойств, соответствующих этим точкам. Таким образом формируется *анимационный клип* (animation clip*)*. При переходе в режим «проигрывания сцены» эти анимации

включаются и воспроизводятся в цикле. В Unity реализована и возможность абстрагирования анимаций. Клипы можно сгруппировать и выделить в отдельные сущности, чтобы в дальнейшем применять их к другим объектам.

Это удобный механизм, но он не позволяет описать все необходимые анимации. Поэтому в Unity есть механизм *контроллеров анимации* (animation controllers). Он построен на конечных автоматах, описывающих переключения между поведениями контролируемого объекта, в том числе, переключения между клипами анимации при смене состояний.

Также для программирования динамики сцены, охватывающей множество объектов, применяют описания на уровне управлений физическими процессами, см. например модель программирования управления космическими аппаратами [10] и среду SimInTech [11, 12].

Нередко в симуляторах и играх применяется модель Entity-Component-System [8]. Эта модель подразумевает связывание «сущностей» с «компонентами» и реализацию динамических процессов в виде «систем».

Часто применяется и модель *потоков данных* (data flow), где поток данных определяет динамику системы (см. пример такого подхода в системе программирования SciVi [13]).

Среди инструментов визуального программирования можно обнаружить и гибридный подход: когда элементы динамики можно задавать и для отдельных объектов, и для их систем. Так, в основе системы GDevelop [14] лежат «*события*» («GDevelop events»), программируя обработку которых пользователь настраивает различные элементы поведения сцены. Программирование в графическом интерфейсе сводится к созданию записей в таблице обработки этих событий, рисунок 2.

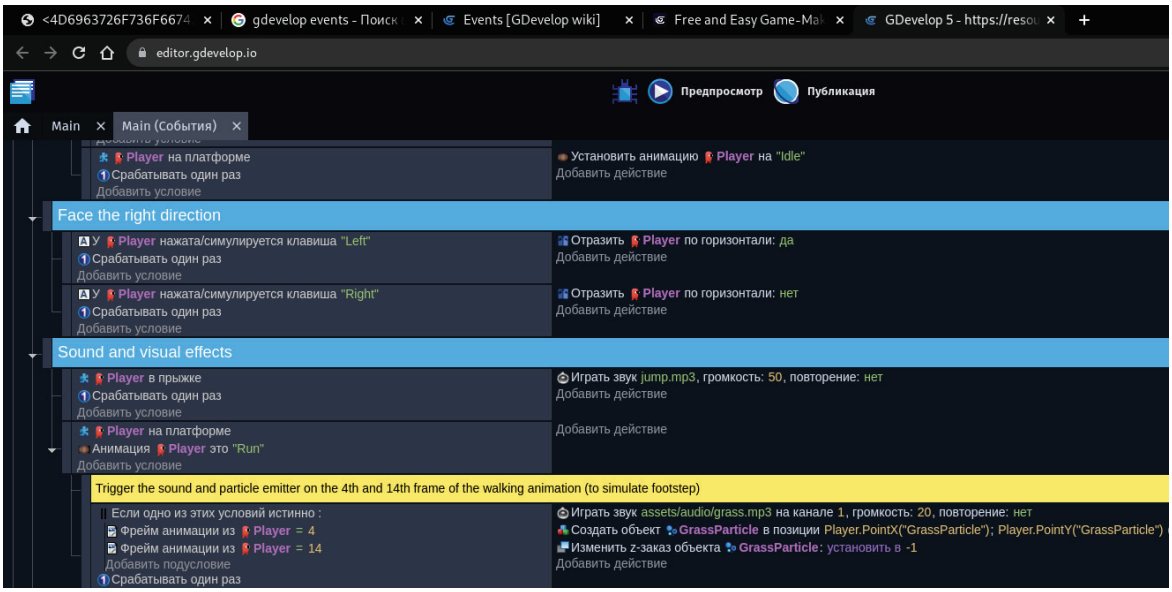

Рисунок 2 – Внешний вид таблицы событий в среде GDevelop

В таблице два столбца – событие и ответное действие. События могут быть как элементарными: старт программы, нажатие на кнопку, и тому подобными, так и возникающими при выполнении сложных условий: «положение персонажа совпало с положением монеты». Действия определяются списками императивных конструкций, таких как «увеличить значение переменной *счёт* на *10*» или «удалить монету из сцены». В таблицу можно вставить и безусловные действия, в том числе, записанные на языке JavaScript. Сформированная таблица обрабатывается построчно сверху вниз 60 раз в секунду.

Особо отметим следующие особенности GDevelop. **1)** У действий и условий могут быть параметры, описываемые выражениями, которые могут содержать вызовы сложных функций. Эти выражения помогают конструировать окна-помощники (wizards). **2)** Можно создавать расширения (см. далее раздел о расширениях). **3)** Описание поведения можно прикрепить не только к одному объекту, но и к группе объектов, заданных некоторым условием, например все объекты данного прокси-класса (см. раздел о модификаторах).

Таким образом, мы видим некоторый спектр подходов к описанию динамики сцены с разношёрстной онтологией, но дающих пользователю похожие возможности. Мы считаем это признаком наличия более фундаментальной модели программирования динамики сцены, которой может быть один из вариантов *исчисления процессов* (process calculus): взаимодействующие последовательные процессы [15] или пи-исчисление [16]. Мы работаем над удобным визуальным интерфейсом конструирования таких процессов. Также возможно, что в решении проблем программирования динамики сцен могут помочь идеи *автоматного программирования*, широко освещённые научным сообществом России.

6. Расширения

Расширения (addons, extensions, plugins, modules, packages) позволяют выносить части функциональности программы во внешние коды, которые пользователь может включать в работу по мере необходимости. Расширения часто применяют в различных программах, в том числе, в системах визуализации, см. например QGIS plugins [17] и GDevelop extensions [18].

Решение о подключении расширения принимается пользователем, см. пример на рисунке 3. Процесс подключения для пользователя происходит в некотором контексте: например в контексте всей системы, или в контексте текущего проекта. Когда пользователь «подключил» расширение, дальнейшее взаимодействие с ним осуществляется по следующей модели.

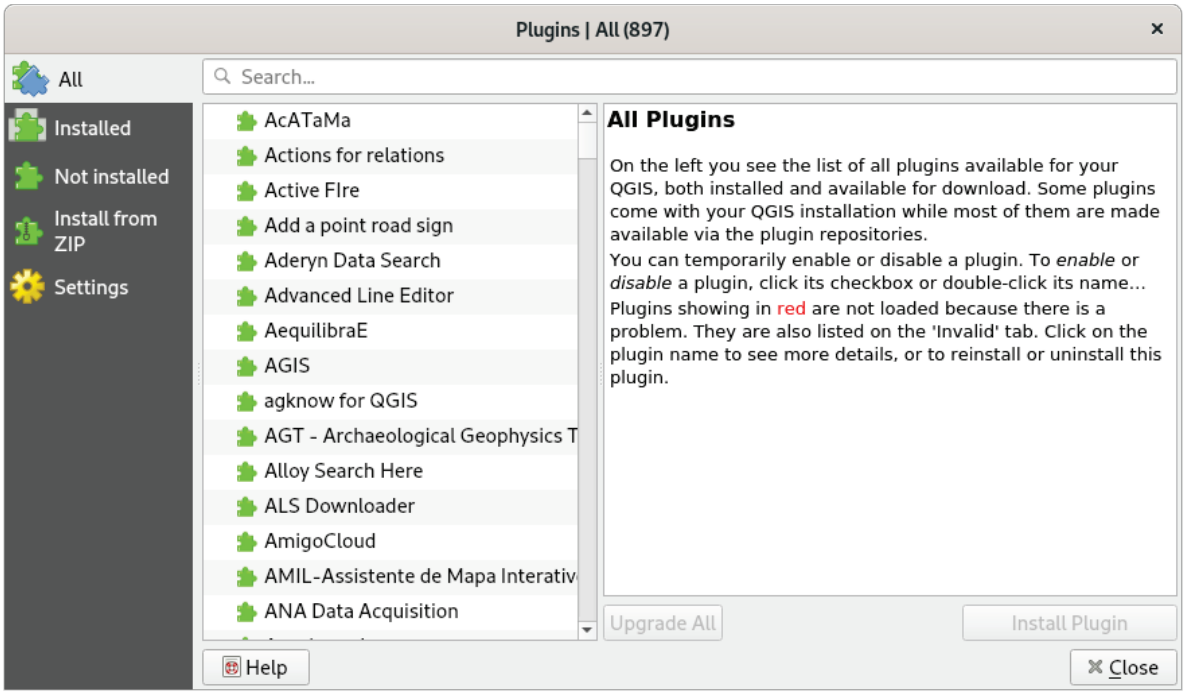

Рисунок 3 – Подключение расширения в среде визуализации ГИС-данных QGIS. Подключение производится в контексте системы, т.е. «активированные» расширения действуют во всех проектах пользователя. QGIS предлагает к установке расширения из своего списка (репозитория). Дополнительно можно добавить другие репозитории, а также установить расширение из архива (меню Install from ZIP)

Система, как правило, передаёт этому расширению управление, предоставляя некоторый интерфейс взаимодействия (API). Расширение при подключении, получив управление и точку входа в API, может добавлять записи в таблицы «возможностей» системы – например, в таблицы доступных пользователю модификаторов.

Иногда API позволяет расширению менять графический интерфейс системы. Пользуясь этим, расширение может, например, напрямую добавлять пункты меню графического интерфейса.

Иногда API ограничен только запросом у расширения таблицы значений в определённом формате, который дальше интерпретируется системой согласно её алгоритмам. Когда пользователь применяет реализованные в подключённом расширении сущности к объектам создаваемого цифрового артефакта, они получают управление и взаимодействуют с объектами в рамках определённого для этих сущностей интерфейса.

В результате взаимодействия на этапе подключения расширения система становится «мощнее». В ней появляются дополнительные виды объектов, модификаторов, элементов динамики сцены и т.п. Таким образом реализуется принцип *аддитивности*, описанный в [5].

Обычно, для подключения расширения пользователь выбирает его из списка. Эти списки могут формироваться различными способами. Список может быть составлен поставщиком программного продукта. Поставщик в некоторых случаях может формировать этот список на основе заявок производителей расширений, реализуя «магазин расширений». Например: магазин Unity Asset Store, который распространяет сущности (assets) разных видов, упакованные в пакеты (packages), которые и являются расширениями.

Иногда пользователю предоставляется возможность вести свой список расширений (список “любимых” расширений) в контексте профиля пользователя (возможно, облачном), и уже расширения из этого списка подключать в контексте системы или текущего проекта. Такой подход реализован, например, в среде Google Tag Manager.

Иногда пользователь может влиять на этот список через каталоги локальной файловой системы, или через файл конфигурации, или через локальные настройки, и т.п. Например, в QGIS пользователь может разместить программный код расширения в файловой системе по определённому пути каталога пользовательских расширений, и тогда оно появляется в графическом интерфейсе в списке доступных для подключения. Также в QGIS реализована возможность «импортировать» расширения из zip-архива. При импорте архив расширения распаковывается в каталог пользовательских расширений.

Отметим, что разработчик системы имеет возможность подключить расширение на уровне исходного кода системы или проекта. Тогда пользователю не нужно инициировать подключение расширения – решение о его подключении уже принято разработчиком и все необходимые действия для подключения обеспечены на уровне программных кодов. Отталкиваясь от этого наблюдения можно предположить, что в идеальном дизайне системы с расширениями все виды сущностей системы: визуальные объекты, модификаторы и пр. – реализованы в расширениях, некоторые из которых подключаются на уровне исходных кодов системы, а некоторые в процессе работы через интерфейс на усмотрение пользователя.

Обратим внимание на интересный подход. В пользовательском интерфейсе некоторых систем отображается функциональность не подключённых расширений. При обращении к ней, пользователь получает сообщение о реализующем эту функциональность расширении и предложение его подключить. Например, в среде GDevelop в пользовательском интерфейсе так «рекламируют» доступные модификаторы.

Мы считаем что расширения являются аналогами модификаторов на другом семантическом уровне. Модификаторы применяются к объектам сцены, а расширения – к проектам или к рабочей среде пользователя, к системе. Модификаторы влияют на поведение объектов сцены, а расширения – на поведение проекта или системы в целом, предоставляя новые возможности или влияя на уже присутствующие в системе.

Как и модификаторы, расширения иногда разделяют на группы по назначению. Процессы подключения для разных групп могут иметь разные пользовательские интерфейсы.

7. Источники данных, визуальные образы, фильтры

Системы визуализации, как правило, визуализируют «данные» (представленные в модели файлов, потоков данных, процессов, сетевых ресурсов и т.п.).

Многие системы научной визуализации выделяют понятие источника данных, см. например системы Visit и Kepler.gl (рисунок 4). Эти системы предлагают пользователю такую модель: “добавить” в проект “данные”, а затем добавить в проект один или более методов визуализации этих данных, и таким образом сформировать вид отображения.

Пользователь добавляет данные указанием пути к ним в локальной файловой системе, или перетаскиванием (drag-and-drop) в графической оболочке операционной системы, или указанием URL из облака либо из другого сетевого ресурса, или, в общем случае, указывая произвольный предварительно сформированный набор данных с некоторым предопределённым протоколом доступа к ним. Некоторые системы позволяют группировать наборы файлов по шаблонам имён файлов. Некоторые участки имён, описываемые такими шаблонами, система может интерпретировать как значения параметров загружаемых объектов (например, времени) и использовать их в построении сцены и её динамики (например, предлагать варианты анимации).

После добавления данных тем или иным образом в проекте появляется артефакт данных вместе с его связью с источником этих данных. В простейшем случае данные целиком копируются и становятся неотъемлемой частью проекта.

Данные могут быть различных типов – например треугольные меши, табличные данные, скалярные и векторные поля, и другие. Если загружен не один артефакт, а набор, то где-либо в пользовательском интерфейсе появляется возможность выбора текущего элемента набора.

При добавлении метода визуализации система может учитывать, к каким данным возможно его применение. Например, в системе Visit есть понятие *текущие данные* – это выбранный сейчас артефакт данных из списка добавленных ранее в проект. При добавлении метода визуализации в списке доступных методов отображаются только совместимые с типом текущих данных. Когда пользователь подтверждает добавление метода визуализации, фиксируется и связь этого метода с текущим артефактом в наборе данных.

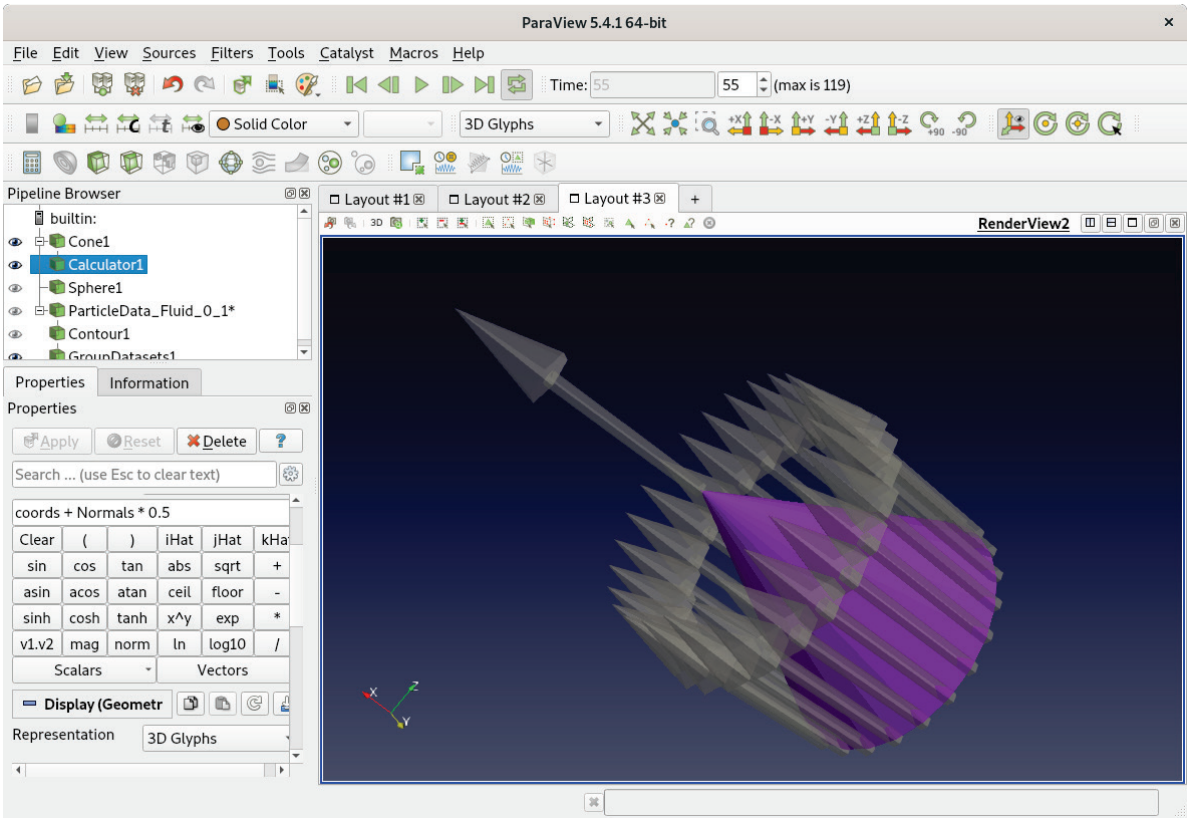

Рисунок 4 – Окно проекта Paraview, в который добавлен конус и добавлен расчётный блок типа Calculator. Блок генерирует вектора на основе опорных точек этого конуса. Активирован «встроенный» в блок метод визуализации векторов

Возможен и дуальный подход. Например, система Kepler.gl позволяет добавлять в проект произвольный (из списка) метод визуализации, а уже после устанавливать его связь с элементами списка данных, добавленных в проект. При установке связи выбрать можно только совместимые с настраиваемым методом визуализации элементы данных.

Некоторые системы позволяют проводить расчёты над добавленными в проект данными. Расчёты могут производится в командном режиме: по завершению такого расчёта, его результат сохраняется в проекте как новый элемент данных. Расчёты могут проводится и в (мы предлагаем термин) *процессном* режиме. В этом случае описывающий расчётный блок объект сохраняется в проекте и доступен для донастройки и последующего перерасчёта. Результат таких расчётов доступен в виде специального артефакта данных — *выходного параметра* этого расчётного блока. Такие блоки иногда называют *фильтрами*.

Выходные параметры блоков могут быть использованы как источники данных для других блоков — как *входные параметры*. Стыкуя блоки по входам и выходам, можно формировать *конвейер* из блоков преобразований (pipeline). Конвейеры могут реагировать на изменение параметров, в том числе и параметров промежуточных звеньев, и автоматически производить перерасчёт с новыми версиями данных.

Реализуют и гибридные модели расчётов, когда пользователю предоставляется возможность как провести однократный расчёт в командном режиме, так и построить сохраняемый в проекте конвейер из настроенных блоков, который пригоден для повторяющихся расчётов. Такую гибридную модель предлагает, например, QGIS processing toolbox.

В результате расчётов формируются новые данные, которые необходимо визуализировать. Некоторые системы учитывают такие новые данные также, как и добавленные пользователями в проект, — позволяют связать эти данные с выбранным методом визуализации. В других системах расчётные блоки оснащают встроенными средствами визуализации, и тогда результатом работы блока являются как данные, так и визуальные образы, которые можно «включить», см. например рис. 4.

Применение серии расчётных блоков и последующее соединение с ними методов визуализации составляет собой, на наш взгляд, существенную часть пользовательского программирования в универсальных системах визуализации.

8. Заключение

В работе приведён обзор некоторых элементов логических моделей пользовательских интерфейсов универсальных систем научной визуализации. Мы считаем эти элементы эффективными, предоставляющими широкий спектр возможностей формирования визуальных образов. Косвенно это подтверждается тем, что они реализованы во многих системах визуализации. Их применение возможно как в новых универсальных системах визуализации, так и по мере необходимости в специализированных системах. Мы экспериментируем с этими моделями, разрабатывая библиотеку программирования систем научной визуализации [21,22].

Взаимодействуя с системой визуализации, пользователь действует в логической модели, предложенной ему. Её ясность, выразительность и эффективность есть ключ к успеху.

9. Благодарность

Авторы благодарят коллег – математиков, физиков, специалистов в области визуализации, а также рецензентов – взаимодействие с которыми привело к появлению данной работы.

10. Список источников

[1] Авербух В.Л. и др., Разработка видов отображения в специализированных системах компьютерной визуализации // Труды 12 Международной конференции по компьютерной графике и машинному зрению ГрафиКон'2002. Нижний Новгород, 16-21 сентября 2002 г., Нижегородский Гос. Университет. – С. 184-189. URL: http://www.cv.imm.uran.ru/e/3546 (дата обращения 30.06.2022).

[2] Авербух В.Л., Компьютерные метафоры и проектирование систем научной визуализации // Труды Второй международной конференции «Трехмерная визуализация научной, технической и социальной реальности. Технологии высокополигонального

моделирования» 24-26 ноября 2010 г. УдГУ, Ижевск. Том 1. Стр. 8-11. URL: https://www.cv.imm.uran.ru/e/915399 (дата обращения 30.06.2022).
[3] Авербух В.Л., Компьютерная визуализация как самостоятельная дисциплина // Труды международной научной конференции «Системы виртуального окружения для комплексной безопасности и антитеррористической защищенности зданий и сооружений» (VRTerro2011). (Сборник посвящается 70-летию Станислава Клименко и Бориса Долговесова). Москва-Протвино. Институт физико-технической информатики. 2011. Стр. 140-145. URL: https://www.cv.imm.uran.ru/e/3199901 (дата обращения 30.06.2022).
[4] Abelson, H.; Sussman, G. J. & Julie Sussman (1996), Structure and Interpretation of Computer Programs, MIT Press/McGraw-Hill, Cambridge.
[5] Abelson, H.; Sussman, G. J. & Julie Sussman, et al, Structure and Interpretation of Computer Programs, JavaScript Edition. URL: https://sourceacademy.org/sicpjs/1.1 (дата обр. 15.10.2022)
[6] Среда разработки Unity. URL: https://unity.com (дата обращения 30.06.2022).
[7] Martin, Adam. "Entity Systems are the Future of MMOG Development". URL: http://t-machine.org/index.php/2007/09/03/entity-systems-are-the-future-of-mmog-development-part-1/ (дата обращения 30.06.2022).
[8] Entity-component-system, URL: https://en.wikipedia.org/wiki/Entity_component_system (дата обращения 30.06.2022).
[9] High-order components в проекте A-frame: https://aframe.io/docs/1.3.0/introduction/entity-component-system.html#higher-order-components (дата обращения 30.06.2022).
[10] А. А. Тюгашев / Подход к обеспечению отказоустойчивости космических аппаратов на основе автоматизации проектирования интеллектуальных бортовых программных средств // Надежность и качество сложных систем. – 2016. – № 4 (16). – С. 106–112. DOI 10.21685/2307-4205-2016-4-15.
[11] Myznikova V.A., Ustimenko V.V., Chubar A.V., Solopko I.V. Mathematical modeling of an unmanned object motion control system in SimInTech environment // Spacecrafts & Technologies, 2022, vol. 6, no. 1, pp. 45-54. doi: 10.26732/j.st.2022.1.06.
[12] Набор медиа материалов по среде SimInTech, URL: https://simintech.ru/science/publications/# (дата обращения 30.06.2022).
[13] Рябинин К. В., Баранов Д. А., Белоусов К. И. Интеграция информационной системы Семограф и визуализатора SciVi для решения задач экспертного анализа языкового контента // Научная визуализация. М.: Национальный исследовательский ядерный университет МИФИ. 2017. Т. 9, № 4. С. 67–77. DOI 10.26583/sv.9.4.07.
[14] Среда разработки GDevelop. URL: https://gdevelop.io (дата обращения 30.06.2022).
[15] C. A. R. Hoare, Communicating Sequential Processes, May 18, 2015 URL: http://www.usingcsp.com/cspbook.pdf (дата обращения 30.06.2022).
[16] Robin Milner, Communicating and Mobile Systems: The Pi Calculus, Cambridge University Press, ISBN 0521643201. 1999. URL: https://en.wikipedia.org/wiki/%CE%A0-calculus (дата обращения 30.06.2022).
[17] Модель плагинов Qgis, URL: https://plugins.qgis.org/ (дата обращения 30.06.2022).
[18] Модель плагинов Gdevelop, URL: https://github.com/GDevelopApp/GDevelop-extensions (дата обращения 30.06.2022).
[19] Васёв П.А., Кумков С.С., Шмаков Е.Ю., О подсистеме истории в среде научной визуализации SharpEye // Труды 23-ей Международной Конференции по Компьютерной Графике и Зрению ГрафиКон’2013. 16–20 сентября, 2013. Владивосток. Стр. 174-177. URL: https://m.cv.imm.uran.ru/e/3040965 (дата обращения 30.06.2022).
[20] Бахтерев М.О., Васёв П.А., Один метод визуализации результатов суперкомпьютерных вычислений // Материалы III Всероссийской научно-технической конференции "Суперкомпьютерные технологии" (СКТ-2014), 29 сентября - 4 октября 2014. Таганрог, НИИ МВС ЮФУ. С. 50-55. URL: https://www.cv.imm.uran.ru/e/3241501 (д. обр. 30.06.2022).
[21] М. О. Бахтерев, П. А. Васёв, Конструктор специализированных конструкторов визуализации // Тезисы XVIII Международной конференции «Супервычисления и математическое моделирование», 23-26 мая 2022 г., РФЯЦ-ВНИИЭФ, г. Саров. URL: https://www.cv.imm.uran.ru/e/3241824 (дата обращения 30.06.2022).
[22] Проект Viewzavr, URL: https://github.com/viewzavr/ (дата обращения 30.06.2022).