DOI: 10.15514/ISPRAS-2023-35(2)-12

Объектно-ориентированный подход к поддержке сценариев в системах оптического моделирования

М.С. Копылов, ORCID: 0000-0002-9526-0766 <dvaag@hotmail.com>
Н.Б. Дерябин, ORCID: 0000-0003-1248-6047 <dek@gin.keldysh.ru>
Е.Ю. Денисов, ORCID: 0000-0002-0614-9100 <eed@spp.keldysh.ru>

Институт прикладной математики им. М.В. Келдыша РАН,
125047, Россия, г. Москва, Миусская пл., д. 4.

Аннотация. В статье рассматриваются особенности поддержки сценариев на языке Python в активно развивающейся системе интерактивной графики. Подобная поддержка является трудоёмкой задачей, которую сложно автоматизировать в общем случае. В качестве решения этой проблемы предлагается подход, позволяющий разработчикам совмещать создание новых компонентов системы с одновременным встраиванием поддержки сценариев без написания избыточного добавочного кода. Результатом работы является дружественный пользователю объектно-ориентированный API, описывающий все аспекты взаимодействия системы и сценариев. Использующие этот API сценарии могут применяться для автоматизации моделирования, а также для расширения возможностей системы с помощью специальных скриптовых классов. Последнее особо важно, так как оставляет обычным пользователям возможность самостоятельно расширять системы с закрытым исходным кодом.

Ключевые слова: автоматизация моделирования; расширяемость; язык сценариев; графический интерфейс

Для цитирования: Копылов М.С., Дерябин Н.Б., Денисов Е.Ю. Объектно-ориентированный подход к поддержке сценариев в системах оптического моделирования. Труды ИСП РАН, том 35, вып. 2, 2023 г., стр. 169-180. DOI: 10.15514/ISPRAS–2023–35(2)–12

An object-oriented approach to scenario support in optics CAD systems

M.S. Kopylov, ORCID: 0000-0002-9526-0766 <dvaag@hotmail.com>
N.B. Deryabin, ORCID: 0000-0003-1248-6047 <dek@gin.keldysh.ru>
E.Yu. Denisov, ORCID: 0000-0002-0614-9100 <eed@spp.keldysh.ru>

Keldysh Institute of the Applied Mathematics of RAS,
4, Miusskaya Sq. Moscow, 125047, Russia.

Abstract. This article discusses the problems of supporting Python scripts in an actively developing interactive graphics system. Such support is a time-consuming task, which is difficult to automate in the general case. As a solution to this problem, we propose an approach that allows developers to combine the creation of new system components with the simultaneous embedding of scripting support without writing redundant additional code. The result is a user-friendly object-oriented API that describes all aspects of interaction between the system and scripts. Scripts using this API can be used to modeling automation as well as to extend the system with custom extension classes. The latter is especially important as it leaves the ability for ordinary users to extend closed-source systems on their own.

Keywords: modeling automation; extensibility; script language; graphical interface

For citation: Kopylov M.S., Deryabin N.B., Denisov E.Yu. An object-oriented approach to scenario support in optics CAD systems. Trudy ISP RAN/Proc. ISP RAS, vol. 35, issue 2, 2023. pp. 169-180 (in Russian). DOI: 10.15514/ISPRAS-2023-35(2)-12

1. Введение

Большинство современных систем оптического моделирования являются весьма сложными программными продуктами, состоящими из множества компонентов, модулей и приложений, использующихся для решения различных практических задач. Чтобы максимально использовать возможности таких систем, пользователям часто приходится перемещаться по слоям меню и диалоговых окон, например, для настройки различных параметров моделирования, счёт которых может идти на десятки, а то и сотни. В итоге наступает момент, когда базовых возможностей, предоставляемых графическим интерфейсом пользователя, становится недостаточно для эффективной работы с системой. Наиболее популярным способом преодоления подобной ограниченности является использование сценариев. С их помощью можно добиться следующих преимуществ.

- Сценарии позволяют автоматизировать процессы моделирования, такие как создание моделей, анализ их поведения, просмотр и изменение параметров модели и т.д. Это позволяет сократить время, затрачиваемое на разработку и обновление моделей, и существенно повысить эффективность работы с системами интерактивной графики.
- Используя языки сценариев высокого уровня и соответствующую системную поддержку, можно расширять возможности систем оптического моделирования путём создания пользовательских приложений, классов, подключаемых модулей и настраиваемых элементов графического интерфейса.

Надо отметить, что внедрение и последующая поддержка сценариев требует существенных усилий со стороны разработчиков системы. Поэтому весьма актуальным является вопрос поиска способов, позволяющих минимизировать подобные усилия. В данной работе предлагается опробованный на практике подход, основанный на использовании специального объектно-ориентированного интерфейса, который встраивается в цепочку базовых классов приложения и предоставляет всю необходимую инфраструктуру для дальнейшего расширения системы.

2. Анализ существующих решений

Можно выделить следующие общие черты, присущие многим ныне существующим системам оптического моделирования, в контексте поддержки сценариев.

- Возможность выполнения определённых действий из командной строки, что может использоваться для автоматизации часто повторяющихся задач. Сценарии в этом случае пишутся на языке командного процессора CMD, PowerShell в среде Windows или Bash в среде Linux и содержат последовательности команд, дополненные необходимыми аргументами. Множество всех возможных команд, как правило, описывается в документации к конкретной системе. Подобные сценарии не имеют большой гибкости, однако их использование является для пользователя достаточно лёгкой задачей, не требующей от него какой-либо существенной квалификации.
- Наличие поддержки одного или нескольких универсальных языков высокого уровня, например, Python или VBScript. Данная поддержка подразумевает наличие в системе специального модуля, называемого интерпретатором сценариев. Одновременно с этим имеется определённый программный интерфейс (API), в рамках которого происходит взаимодействие между сценариями и различными компонентами системы. Сценарии на языке высокого уровня являются гораздо более гибкими, но при этом от пользователя требуются, как минимум, базовые знания использующегося языка и задействованного API.

- Присутствие специальных инструментальных средств для редактирования и отладки сценариев. Это может быть как простой текстовый редактор, так и продвинутый редактор сценариев, поддерживающий дополнительные функции, например, отладку во время выполнения, использование точек останова, подсветку синтаксиса и т.д.

Рассмотрим особенности поддержки сценариев в некоторых системах интерактивной графики более подробно.

Система Autodesk Maya имеет поддержку сразу нескольких языков сценариев: MEL (Maya Embedded Language) и Python. Сценарии, написанные на языке MEL, используют процедурный MEL API [1], обладающий очень богатым функционалом. Как пример, на данном языке написана большая часть окружения Maya и сопутствующих расширений. Имеется возможность записи действий в сценарий на MEL, т.е. создание макросов. Среди недостатков данного языка можно отметить отсутствие объектной ориентации и использование собственного синтаксиса, что требует наличия у пользователя определённых навыков. Maya Python API [2, 3] свободен от вышеперечисленных недостатков, однако этот API очень сложен для большинства прикладных задач из-за своего весьма низкого уровня. Сценарии можно создавать, запускать и сохранять непосредственно в Autodesk Maya с помощью встроенного редактора сценариев. В равной мере поддерживается выполнение сценариев командной строки.

Весьма популярный пакет программ для создания трёхмерной компьютерной графики Blender также поддерживает автоматизацию и расширение с помощью языка Python. Для взаимодействия сценариев и различных компонентов системы используется Blender Python API [4]. Запуск сценариев можно осуществлять из встроенного редактора сценариев или из командной строки. Среди недостатков можно отметить то, что доступ к объектам модели осуществляется через наборы коллекций, что, на наш взгляд, является далёким от объектно-ориентированного стиля. Также в Blender отсутствует автоматические преобразования внутренних структур в их аналоги, доступные через интерфейс Blender Python API, что требует от разработчиков этого пакета программ явного дублирования усилий и может приводить к пропущенной функциональности [5].

Аналогичные недостатки присутствуют и в системе трехмерного моделирования FreeCAD, поддерживающей язык Python. Любое расширение внутреннего C++ API этой системы требует ручного создания привязок к Python, что осуществляется с помощью дополнительных заголовочных и xml-файлов [6]. При этом надо отметить, что в некоторых случаях данная проблема частично решается с помощью применения методов автоматической генераций привязок между объектами C++ и объектами того или иного языка сценариев [7, 8, 9]. Отдельно можно выделить библиотеку CLIF [10], позволяющую создавать обёртки для Python с помощью статического анализа C++ кода, что обеспечивает более стабильную и безопасную работу с этим кодом из Python [11].

Система оптического моделирования Rhinoceros 3D включает в себя программный инструмент Rhinoscript, основанный на технологии COM. Этот инструмент используется для создания и запуска сценариев, написанных на языке Microsoft VBScript. Пользователю доступен Rhinoscript API, содержащий более пятисот различных методов, позволяющих легко автоматизировать большинство прикладных задач [12]. Среди недостатков можно отметить отсутствие возможности расширения системы с помощью сценариев. Их запуск из командной строки также не поддерживается, что может затруднять автоматизацию части задач. Дополнительно нужно подчеркнуть, что COM-разработка – это довольно сложное занятие, требующее от разработчиков явного выполнения множества рутинных задач, что усложняет поддержку нового функционала в сценариях по мере развитии системы.

Поддержка сценариев в САПР CATIA также основана на технологии COM. Поддерживается запуск сценариев как из графического интерфейса пользователя, так и из интерфейса командной строки. Особенностью этой САПР является то, что она может быть оснащена

внешним интерпретатором сценариев. В этом случае для разработки сценариев пользователи могут использовать любой язык, поддерживающий технологию COM, например, Python, Java или C#. САПР CATIA поставляется с достаточно функциональным редактором сценариев и имеет большой и хорошо документированный Automation API [13]. Недостаток, связанный с высокой сложностью COM разработки, актуален и для данной системы.

3. Описание подхода

В этом разделе описаны основные возможности нашего подхода и детали его реализации.

3.1 Концепция целевого интерфейса

Разрабатываемая нами система оптического моделирования реализует строгую модульную архитектуру, в которой выделяются две главные группы компонентов: модули вычислительного ядра и модули графического интерфейса пользователя [14]. Важно отметить тот факт, что весь функционал системы сосредоточен во множестве компонентов ядра, и это позволяет использовать его в отрыве от других модулей. Эта особенность делает возможным использование различных клиентских сторон пользовательского интерфейса (frontend), например, консольных или сетевых без глубокой переделки всей системы. В сильно упрощённом виде функциональная схема системы, используемая нами, показана на рис.1.

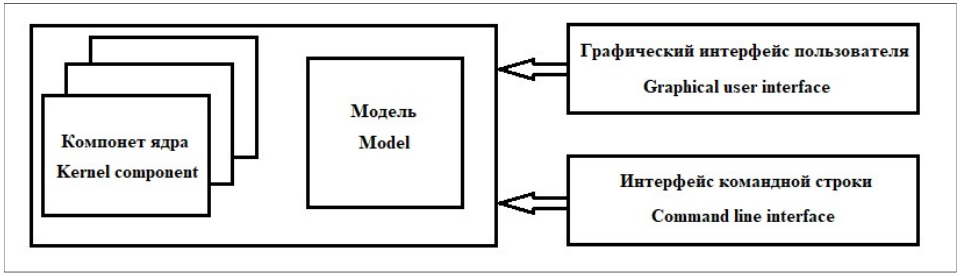

Рис. 1. Функциональная схема системы
Fig. 1. Functional scheme of the system

Для взаимодействия как между собой, так и с загруженной моделью (сценой) все компоненты нашей системы используют так называемый унифицированный целевой интерфейс. Этот интерфейс строится из требований к графическому интерфейсу, т.е. полнота, непротиворечивость и достаточность. Он реализуется двумя дружественными базовыми C++ классами: `Entity` и `EntityType`. Данный интерфейс предоставляет набор операций, позволяющий работать с объектами системы в терминах *свойств* и *действий*.

Под свойствами подразумеваются внутренние атрибуты объекта. Это могут быть как элементарные типы данных, например, скаляры или массивы, так и ссылки на другие объекты системы. Целевой интерфейс предоставляет специальный метод для определения свойства объекта, с помощью которого задаётся имя свойства и тип его значений. Дополнительно целевой интерфейс предоставляет виртуальные методы для чтения и установки значения свойств `SetProperty()` и `GetProperty()`.

Под действиями подразумеваются виртуальные функции (методы) класса, которые могут иметь параметры и возвращать результат `ExecProc()` и `ExecFunc()`. Параметрами и результатами этих функций могут быть как простые объекты, например, элементарные типы данных, так и ссылки на любые объекты системы, реализующие целевой интерфейс. Также целевой интерфейс предоставляет специальный метод для определения действий,

позволяющий задать имя действия, типы параметров, тип результата и метод класса, непосредственно исполняющий каждое конкретное действие.

Все классы производные от `Entity` реализуют функционал объектов системы, а классы производные от `EntityType` реализуют тип объекта. Экземпляров объектов может быть сколь угодно много, однако каждый тип объекта существует в системе в единственном числе и имеет глобальную точку входа (паттерн *singleton*). Классы типов используются для создания новых экземпляров объектов с помощью механизма конструкторов.

3.2 Взаимодействие с Python API

Возможность работы со свойствами и действиями существующих объектов системы исключительно через интерфейс базового класса `Entity` обеспечивает их совместимость как между собой, так и с будущими (ещё не разработанными) объектами и классами. Поэтому поддержка сценариев на языке высокого уровня Python была добавлена в нашу систему с учётом этой ключевой особенности. Интерпретатор языка сценариев Python был встроен в нашу систему с помощью пакета CPython [15], дополненного специальным модулем, реализующим унифицированный целевой интерфейс. По сути, интерпретатор сценариев стал ещё одной клиентской стороной нашей системы, позволяющей выполнять сценарии как из графического интерфейса пользователя, так и из командной строки. Более подробную информацию об интеграции данного языка можно найти в [16].

Благодаря подобной реализации, при расширении системы новыми классами разработчикам более не требуется писать какой-либо дополнительный код для доступа к этим классам из сценариев. Новые классы становятся доступными из сценариев автоматически при условии, что они будут наследовать унифицированный целевой интерфейс.

На листинге 1 приведён пример кода, который определяет новый тип целого числа, хранящего только положительные значения (основной язык C++).

```
PosIntType::PosIntType()
  : EntityType("PosInt")
  {
  Def(new Property(this, "value", &PosInt::m_value, TIntVal,
    READ_ONLY));
  Def(new XAction(this, "Set", &PosInt::Set,
    Arg("value", TIntVal));
  }
```

Листинг 1. Пример определения нового типа
Listing 1. An example of a new type definition

Новый тип наследуется от базового `EntityType`. В его конструкторе определяется свойство `value`, которое будет использоваться для чтения значения, и действие `Set` для его установки. Для создания объекта данного типа определяется функция *конструктор* (основной язык C++), листинг 2.

```
Entity *PosIntType::ConstructObject() const
  {
    return new PosInt();
  }
```

Листинг 2. Функция конструктор
Listing 2. Constructor function

Свойства и действия целевого интерфейса явно преобразуются в конструкции языка Python, поэтому сценарии могут использовать обычную точечную нотацию для доступа к наборам свойств и действий, как показано на листинге 3 (основной язык Python):

```
x = PosInt() # вызов конструктора объекта типа PosInt
x.Set(10) # установка нового значения через вызов действия Set
print(x.value) # чтение значения через обращение к свойству value
```

Листинг 3. Точечная нотация
Listing 3. Dot notation

Такой подход позволяет разрабатывать сценарии с использованием устоявшегося синтаксиса языка Python. В нашем случае нет необходимости использовать какие-либо дополнения или отдельные синтаксические конструкции для взаимодействия с объектами системы. От пользователя требуются лишь базовые знания языка Python.

Другим преимуществом данного подхода является то, что получившийся в итоге интерфейс программирования сценариев (Python API) полностью совпадет с унифицированным целевым интерфейсом системы с точки зрения иерархии объектов, их свойств и действий. На рис. 2 показан пример такой иерархии классов, определяющий различные типы узлов модели (сцены).

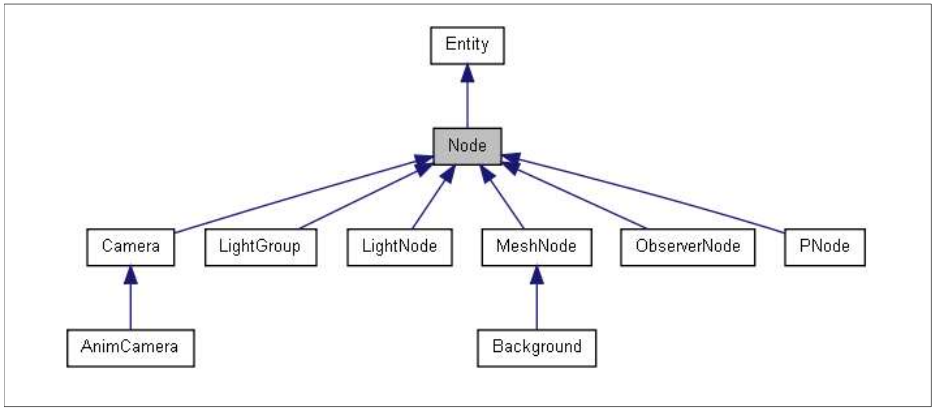

Рис. 2. Иерархия классов узлов модели
Fig. 2. Hierarchy of node classes of the model

Отдельно нужно отметить один важный момент, связанный с тем, что целевой интерфейс из-за использования языка сценариев теперь выходит непосредственно на конечного пользователя. От этого интерфейса требуется быть удобным для человека, т.е. по возможности понятным, естественным и непротиворечивым. Сюда можно включить использование разработчиками понятных имён свойств, действий и типов, использование свойств там, где это возможно, вместо избыточных Get/Set методов и т.д.

3.3. Параметрические объекты

Автоматизация моделирования с помощью сценариев, хранящихся вне самой модели, например, в виде файлов на диске, может быть затруднена, так как изменение параметров модели часто требует изменение самого сценария. Данный недостаток, присущий многим другим системам интерактивной графики, был решён нами с помощью использования специальных параметрических объектов, которые, с одной стороны, являются частями модели, а с другой стороны, содержат в себе сценарий на языке Python, представляющий из себя класс расширения.

С технической точки зрения параметрические объекты ничем не отличается от других объектов системы, реализующих унифицированный целевой интерфейс. Однако подобные объекты имеют среди своих свойств специальное скрытое поле, содержащее программный код класса расширения. В отличие от обычных сценариев, являющихся набором последовательно выполняющихся подпрограмм, сценарии классов расширений не имеют явной точки входа, с которой начинается их выполнение. Вместо этого для взаимодействия с

остальной системой используются функции обратного вызова, которые вызываются ядром комплекса оптического моделирования. Классы расширений позволяют задавать, а затем многократно изменять параметры сценарного объекта без изменения кода самого класса расширения.

Классы расширений полностью поддерживают парадигму объектно-ориентированного программирования, т.е. объектную ориентацию, наследование и полиморфизм, тем самым предоставляя пользователю удобную возможность расширения конечной системы новым функционалом.

На листинге 4 приведён пример кода, который определяет новый класс расширения, описывающий тип геометрического объекта в форме параллелепипеда (основной язык Python):

```
class Box(PMesh):
    # Definition of parameters
    org = Param("Origin", VectType(Size))
    size = Param("Size", VectType(Size(0.001)), (300, 200, 100))
    # Creation method
    def Init(self):
        # Create parts
        self.AddPart("Side")
    # Evaluation method
    def Eval(self):
        # Calculate box boundaries
        x1, y1, z1 = self.org
        length, width, height = self.size
        x1 -= length / 2; y1 -= width  / 2; z1 -= height / 2
        x2 = x1 + length; y2 = y1 + width; z2 = z1 + height
        # Reconstruct box geometry
        self.ClearGeometry()
        self.AddVerts(0, [[x1, x2]], [[y1, y2]], [[z1, z2]])
        #...
        self.AddVerts(5, [[x1]], [[y1, y2]], [[z1], [z2]])
    # Notification handler
    def OnNotify(self, object, reason):
        pass
```

Листинг 4. Определение нового класса расширения
Listing 4. Defining a new extension class

Структура класса расширения состоит из его имени и последующих определений методов и параметров. Параметры задаются через вызов специального метода `Param()`. Ими могут быть любые типы, реализующие целевой интерфейс. Дополнительно параметру можно задать значение по умолчанию. Специальный метод `Init()` исполняет роль конструктора и вызывается однократно при создании нового экземпляра объекта. Метод `Eval()` автоматически вызывается ядром системы при изменении параметров класса. Назначением этого метода является обновление представления объекта в соответствии с новыми значениями параметров.

С точки зрения целевого интерфейса, параметры класса расширения являются свойствами, а методы, соответственно, являются действиями. Это означает, что доступ к ним возможен из любой части системы, например, из графического интерфейса или из сценариев, что демонстрируется на рис. 3.

Так как параметрические объекты являются частями модели (сцены), должна обеспечиваться их корректная работа при загрузке модели в более старую или более новую версию комплекса

оптического моделирования. Это достигается с помощью механизма проверки версий, который гарантирует, что код класса расширения сможет функционировать должным образом.

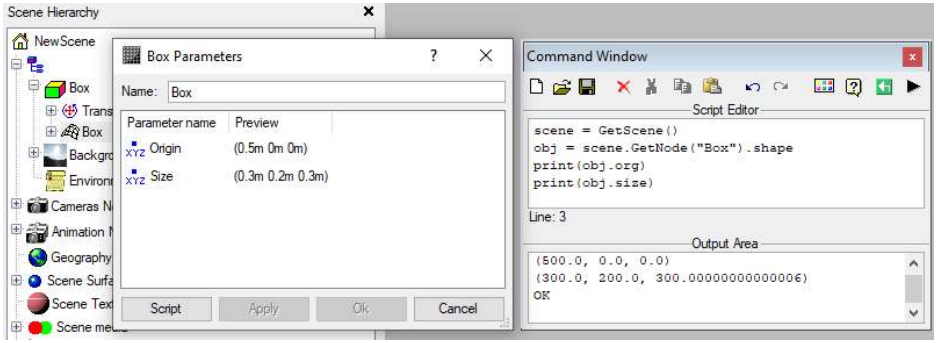

Рис. 3. Различные способы доступа к параметрам объекта
Fig. 3. Different ways of accessing object parameters

Важной особенностью параметрических объектов является возможность реагирования на различные события, происходящие в других объектах системы, например, на изменение их состояния. Эта возможность реализуется благодаря специальному механизму уведомлений, описываемому ниже.

3.4. Механизм сообщений об изменении объектов

Целевой интерфейс нашей системы поддерживает механизм уведомлений об изменениях состояния объектов. Данный механизм базируется на *ссылках*, которые представляют собой разновидность умного указателя и имеют одну важную дополнительную особенность: они являются двухсторонними. То есть хранят указатель не только на ссылаемый объект, но и обратный указатель на объект, содержащий ссылку [17].

Каждый объект системы может уведомить об изменении своего состояния с помощью вызова метода `Entity::NotifyChange()` целевого интерфейса. Результатом этого будет вызов виртуального метода `Entity::OnChange()` во всех объектах системы, которые владеют ссылками на объект, уведомивший об изменении.

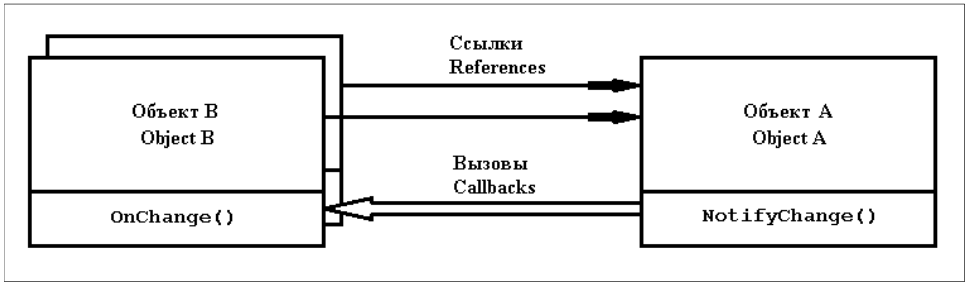

Рис. 4. Уведомление об изменении
Fig. 4. Change notification

Реализация метода `OnChange()` в объекте владельце должна проанализировать причину изменений, выполнить соответствующие действия, а затем сообщить завершена ли обработка изменения или её следует продолжить, т.е. передать обработку изменения следующему объекту из списка объектов владельцев ссылки (рис. 4).

Аргументами вызова `NotifyChange()` являются причина уведомления, которая задаётся в каждом конкретном классе в виде констант перечисления, и любые дополнительные параметры, которые могут быть полезны при обработке уведомления в методе `OnChange()` объекта владельца.

Данная функциональность целевого интерфейса была воплощена и в классах расширений. Для этого в каждом экземпляре параметрического объекта имеется специальное скрытое поле, хранящее список ссылок на отслеживаемые объекты. Добавление конкретного объекта в этот список осуществляется с помощью метода `EnableNotify()` получающего объект в качестве аргумента. Как правило, данный метод вызывается в конструкторе класса расширения, но явных ограничений здесь нет, подписаться на приём уведомлений можно в любой момент времени и из любого места сценария. Обработчик уведомлений реализуется в методе `OnNotify()`, который является непосредственной надстройкой над методом `OnChange()` целевого интерфейса.

На листинге 5 приведён пример кода класса расширения, реализующего обработчик, отслеживающий изменение расположения интересуемого узла сцены (основной язык Python):

```
# Notification handler
def OnNotify(self, obj, reason):
    # Check the object that issued the notification
    s = GetScene()
    if ((s != None) and (obj == s.GetNode("Box"))):
        # Check reason
        if ((reason is not None) and (reason ==
        NodeNotifyReason.TRANSFORM_CHANGED)):
            # Position of 'Box' node was changed, do some stuff
            ...
```

Листинг 5. Пример кода класса расширения, реализующего обработчик
Listing 5. Example code for an extension class that implements a handler

Параметрические объекты, использующие механизм уведомлений, позволяют отказаться от необходимости опроса состояния отслеживаемых объектов, что существенно повышает эффективность процессов моделирования и экономит системные ресурсы.

4. Результаты

Описанная технология была успешно внедрена в разрабатываемый нами программный комплекс оптического моделирования и реалистичной компьютерной графики Lumicept. Одним из главных преимуществ данной технологии стала возможность расширения функциональности системы без непосредственного участия разработчиков, т.е. силами конечных пользователей, что позволяет выполнять точную настройку и подгонку комплекса под их нужды, особенно в случае необходимости добавления моделирования вычислений тех или иных стандартов и метрик. Это весьма важно в свете того, что различные архитектурно-проектировочные или оптико-дизайнерские компании, как правило, используют различные стандарты освещённости или яркости, поэтому использование сценариев и классов расширений решает сразу несколько проблем.

- Экономическая эффективность. Доработка кода программного комплекса для каждого конкретного заказчика может быть весьма дорогостоящей. Благодаря возможности добавлять расчеты с помощью сценариев, пользователи могут настраивать комплекс в соответствии со своими конкретными потребностями, не требуя внесения изменений в его исходный код. Это позволяет им сэкономить как время, так и деньги.
- Гибкость. Использование сценариев обеспечивает большую гибкость при добавлении новых функций и возможностей. Это связано с тем, что сценарии можно легко изменять

и обновлять, а от пользователей не требуются продвинутые навыки и знания в области программирования.

- Безопасность. Изменение кода программного комплекса для каждого заказчика не только дорого, но и небезопасно. Существует высокая вероятность внесения ошибок, которые могут вызвать серьезные проблемы. Сценарии и классы расширений обеспечивают более безопасный подход к настройке.
- Повышенная масштабируемость. Сценарии облегчают интеграцию пользовательских объектов и функций в специальные библиотеки, которые впоследствии можно легко расширять по мере возрастания потребностей.

К настоящему времени для нашего комплекса был создан набор из ~200 сценарных расширений, большая часть которых представляет собой классы объектов сцены пользовательских типов, таких как узлы, геометрии, микрогеометрии, а также функции. Оставшаяся часть является различными видами симуляторов.

4.1. Примеры расширений

Одним из практических примеров расширения функциональности нашего программного комплекса является сценарный симулятор определения уровня дискомфорта, вызванного бликами от источников света (Glare Rating). Данный симулятор позволяет моделировать естественное освещение пространства с использованием различных параметров, задаваемых пользователем, и последующим сохранением результатов в файловой системе.

Area index	Solid Angle (sr)	Integral Luminance (cd/m2*sr)	Average Luminance (cd/m2)
1	0.007903849	4.521654	572.0826
2	0.004634725	2.102725	453.6891
3	0.002781498	0.945254	339.8364
4	0.002663621	0.902460	338.8093
5	0.002257292	1.260026	558.2027
Sum:	0.02024098	9.732119	480.8125

Рис. 5. Симулятор для анализа бликов
Fig. 5. Simulator for glare rating analysis

Основными параметрами симулятора являются: ориентация сцены, положение наблюдателя, яркость источников освещения. Множество дополнительных параметров позволяет

управлять различными характеристиками моделирования, от выбора метода расчета – BRT (Backward Ray Tracing) или PT (Path Tracing) до формата имен генерируемых файлов с результатами.

Результатом работы симулятора является изображение, в котором области яркости для заранее заданного интервала выделяются изолиниями, а общее значение рейтинга бликов (Unified Glare Rating) и детальная информация по каждой области сохраняется в файл отчёта, как это показано на рис. 5.

Другой пример представляет собой сценарное расширение, позволяющее задать геометрию в виде криволинейной поверхности с синусоидальным профилем в сечении, например, если требуется моделирование поверхностей с такими формами. Для того чтобы добавить подобный объект в модель, пользователю достаточно перетащить его методом drag and drop из библиотеки объектов в сцену, затем настроить необходимые параметры, такие как размеры требуемой поверхности, амплитуда и период кривой, шаг разбиения треугольной сетки, образующей криволинейную поверхность, и т.д., что демонстрируется на рис. 6.

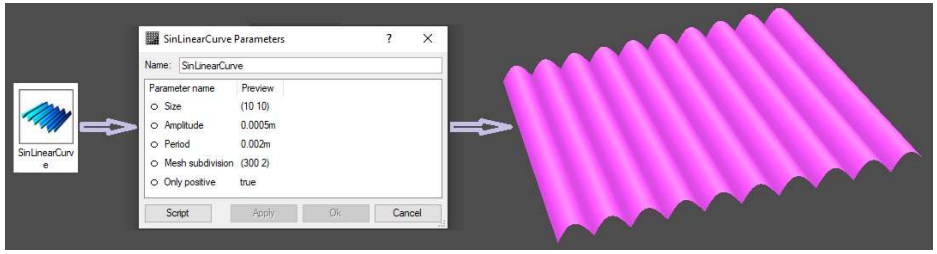

Рис. 6. Параметры сценарного объекта и его отображение в сцене
Fig. 6. Scenario based object parameters and its appearance in the scene

5. Заключение

Описанный подход к поддержке сценариев широко используется в разрабатываемом нами программном комплексе оптического моделирования. Он позволил разработчикам уменьшить количество кода, над которым ведётся единовременная работа в рамках жизненного цикла комплекса, благодаря унификации внутреннего целевого программного интерфейса объектов системы с программным интерфейсом сценариев. Удалось снизить вероятность возникновения ошибок и расхождений функциональности, а также увеличить скорость разработки. Пользователи комплекса, в свою очередь, получили в распоряжение удобный интерфейс сценариев, позволяющий автоматизировать различные прикладные задачи, а также создавать свои собственные эффективные расширения с минимальными затратами времени.

Список литературы / References

[1] PyMEL for Maya, Available at: https://help.autodesk.com/cloudhelp/2018/JPN/Maya-Tech-Docs/PyMel/index.html, accessed 13.02.2023.
[2] Mechtley A., Trowbridge R. Maya Python for Games and Film: A Complete Reference for the Maya Python API. CRC Press, 2011, 381 p.
[3] Collado G.E.G. Modeling and Python Scripting in Maya for the Animation Short Style. Bachelor's Thesis. Department of Computer Engineering, California Polytechnic State University, 2016, 22 p.
[4] Blender 3.4 Python API Documentation, Available at: https://docs.blender.org/api/current/index.html, accessed 13.02.2023.
[5] Anders M. Blender 2.49 Scripting, Packt Publishing, 2010, 282 p.
[6] Machado F., Malpica N., Borromeo S. Parametric CAD modeling for open source scientific hardware: Comparing OpenSCAD and FreeCAD Python scripts. PLoS ONE, vol. 14, issue 12, 2019, article id e0225795, 30 p.

[7] Goyal M., Osborne I., Pivarski J. The Awkward World of Python and C++. arXiv preprint arXiv:2303.02205, 2023, 6 p.
[8] Buse F., Bellmann T. General Purpose Lua Interpreter for Modelica. In Proc. of the 14th Modelica Conference, 2021, pp. 425-431.
[9] Standish R. C++ Reflection for Python Binding. Overload, issue 152, 2019, pp. 11-18.
[10] C++ Language Interface Foundation (CLIF), Available at: https://github.com/google/clif, accessed 30.03.2023.
[11] Can D., Martinez V.R. et al. Pykaldi: A Python Wrapper for Kaldi. In Proc. of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018, pp. 5889-5893.
[12] RhinoScript Guides. Available at: https://developer.rhino3d.com/guides/rhinoscript/, accessed 13.02.2023.
[13] Li Y. Research of Integration Technology between CATIA and TOOLMANAGER Based on CAA, International Journal of Advanced Network, Monitoring and Controls, vol. 1, issue 1, 2016, pp. 130-135.
[14] Барладян Б.Х., Волобой А.Г. и др. Интеграция реалистичной графики в системы автоматизированного проектирования и управления жизненным циклом изделия. Программирование, том 44, вып. 4, 2018 г., стр. 26-35 / Barladian B.K., Voloboy A.G. et al. Integration of realistic computer graphics into computer-aided design and product lifecycle management systems, Programming and Computer Software, vol. 44, issue 4, 2018, pp. 225-232.
[15] Extending and Embedding the Python Interpreter. Available at: http://docs.python.org/3/extending/index.html, accessed 13.02.2023
[16] Дерябин Н.Б., Жданов Д.Д., Соколов В.Г. Внедрение языка сценариев в программные комплексы оптического моделирования. Программирование, том 43, вып. 1, 2017 г., стр. 40-53 / Deryabin N.B., Zhdanov D.D., Sokolov V.G. Embedding the Script Language into Optical Simulation Software // Programming and Computer Software, vol. 43, issue 1, 2017, pp. 13-23.
[17] Галактионов В.А., Дерябин Н.Б., Денисов Е.Ю. Объектно-ориентированный подход к реализации систем компьютерной графики. Информационно-измерительные и управляющие системы, вып. 6, 2009 г., стр. 96-108 / Galaktionov V.A., Deryabin N.B., Denisov E.Yu. Object-oriented approach for computer graphics systems implementation. Information-measuring and Control Systems, issue 6, 2017, pp. 96-108 (in Russian).

Информация об авторах / Information about authors

Михаил Сергеевич КОПЫЛОВ – младший научный сотрудник. Сфера научных интересов: языки программирования, компьютерная графика, вычислительная оптика, трассировка лучей

Mikhail Sergeevich KOPYLOV, junior researcher. Research interests: programming languages, computer graphics, computing optics, ray tracing techniques.

Николай Борисович ДЕРЯБИН – научный сотрудник. Сфера научных интересов: архитектура систем оптического моделирования, компьютерная графика, вычислительная оптика.

Nikolai Borisovich DERYABIN, researcher. Research interests: architecture of optical simulation software systems, computer graphics, computing optics.

Евгений Юрьевич ДЕНИСОВ – научный сотрудник. Сфера научных интересов: компьютерная графика, создание программных систем оптического моделирования, автоматическое тестирование, параллельные и распределённые вычисления.

Evgeniy Yuryevich DENISOV, researcher. Research interests: computer graphics, elaboration of optical simulation software systems, automatic testing, concurrent and distributed computing.