Особенности использования сценариев при автоматизации задач в системах оптического моделирования

М.С. Копылов [1]

[1] ИПМ им. М.В. Келдыша РАН, Миусская пл. 4, Москва, 125047, Россия

Аннотация

В данной статье описаны подходы к использованию сценариев при автоматизации прикладных задач в рамках работы с комплексом оптического моделирования и фотореалистичной компьютерной графики. Комплекс оптического моделирования должен включать в себя поддержку выполнения сценариев, поскольку обычных средств, например предоставляемых с помощью графического интерфейса пользователя, как правило, бывает недостаточно для выполнения многообразных задач, возникающих на практике. Производится краткий обзор существующих решений. Предлагаются методы, обеспечивающие эффективную интеграцию поддержки сценариев в существующую систему оптического моделирования. Данные методы обеспечивают полный доступ из сценариев к модулям системы, включая вычислительные модули, различные модули-симуляторы и т.д. При этом поддерживаются как тривиальные сценарии, содержащие обычные последовательности команд, так и более сложные, использующие возможности скриптовых языков высокого уровня, таких как Python. Отдельно рассматривается специальный высокоуровневый API, который используется для взаимодействия сценариев написанных на языке Python и системы оптического моделирования. Этот API также может использоваться для расширения возможностей системы новыми параметрическими объектами. Подробно рассмотрены интегрированные в систему оптического моделирования специализированные программные модули, такие как интерпретатор пакетных сценариев, интерпретатор и редактор скриптов, редактор классов расширений. Приведены примеры автоматизации моделирования на основе сценариев.

Ключевые слова

Автоматизация моделирования, расширяемость, язык сценариев, командная строка, Python, графический интерфейс.

Usage of Scenario Based Modeling Automation in Optics CAD Systems

M.S. Kopylov [1]

[1] The Keldysh Institute of the Applied Mathematics of RAS, Miusskaya Sq. 4, Moscow, 125047, Russia

Abstract

This article describes approaches to the use of scenarios in the automation of applied tasks within the framework of working with a software complex of optical modeling and photorealistic computer graphics. Optical modeling complex should include support for scripting, since conventional tools, such as those provided using a graphical user interface, are usually not enough to perform a variety of tasks that arise in practice. A brief overview of existing solutions is made. Methods are proposed that ensure effective bringing of scenario support into an existing optical modeling system. These methods provide full access from scripts to system modules, including computational modules, various simulator modules, etc. Both trivial scripts containing ordinary command sequences and more complex ones using the capabilities of high-level scripting languages such as Python are

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: dvaag@hotmail.com (М.С. Копылов)
ORCID: 0000-0002-9526-0766 (М.С. Копылов)

supported. Separately, a special high-level API is considered which is used for the interaction between of scripts written in the Python language and the optical modeling system. This API can also be used to extend the capabilities of the system with new parametric objects. The specialized software modules integrated into the optical modeling system, such as the batch interpreter, script interpreter and editor, extension class editor, are considered in detail. Examples of scenario-based modeling automation are given.

Keywords
Modeling automation, extensibility, script language, command prompt, Python, graphical interface.

1. Введение

Все современные системы оптического моделирования являются достаточно сложными программными продуктами, содержащими в себе различные вычислительные модули и приложения, использующиеся для решения многообразных практических задач. Сложность подобных систем проявляется и в их графическом интерфейсе пользователя. Как правило, он является весьма продвинутым и зачастую очень перегруженным, так как имеется необходимость предоставлять возможности для настройки и отображения различных параметров моделирования, счёт которых может идти на десятки, а то и сотни, например положения источников света в сцене, величины интенсивности этих источников, значения различных критериев модели и т.д.

Разрабатываемая нами система оптического моделирования [1] также не избежала этой участи. В определённый момент времени, возможностей, предоставляемых графическим интерфейсом пользователя, стало не достаточно для обеспечения эффективной работы с системой. Одним из способов преодоления возникшей ограниченности стало использование сценариев. Отметим основные преимущества, которые можно получить с их помощью:

1. Автоматизация моделирования. Пользователь может задать необходимую последовательность действий для конкретной задачи, затем сохранить её в файле сценария и повторно использовать в дальнейшем.
2. Более лёгкое расширение возможностей системы. Используя различные скриптовые языки высокого уровня, пользователь может расширять возможности конечной системы, добавляя в неё новые типы, классы и объекты, не прибегая к непосредственному программированию на языках, требующих от него весьма высокой квалификации, таких как C++ на которых, как правило, разработано большинство систем оптического моделирования. Учитывая, что большинство таких систем являются системами с закрытым исходным кодом, их расширение с помощью сценариев остаётся единственным способом доступным для обычных пользователей.

Надо отметить, что можно выделить существование двух основных классов сценариев различающихся по своей внутренней структуре:

1. Простые сценарии, не содержащие в себе программный код. Сюда можно отнести пакетные файлы командной строки (файлы с расширением bat, cmd в среде Windows и sh в среде Linux). Такие сценарии, как правило, содержат лишь последовательности команд, дополненные необходимыми аргументами. Команды в данных сценариях выполняются по очереди последовательно. Разработка подобных сценариев является весьма простой задачей для пользователя, так как ему достаточно лишь ознакомится со списком доступных команд той или иной системы оптического моделирования.
2. Сценарии, которые содержат в себе программный код. Сюда относятся все сценарии, использующие возможности и конструкции какого-либо языка программирования, такие как циклы, ветвления, классы, функции, процедуры, переменные и т.д. Это могут быть как языки общего назначения, например Python, Java, Perl так и специализированные языки, являющиеся частью конкретных приложений, например AutoIt. Разработка подобных сценариев требует от пользователя как минимум базовых знаний использующегося в них языка программирования.

В нашу систему оптического моделирования была интегрирована поддержка обоих вышеперечисленных классов сценариев. Для реализации этой поддержки, были применены

определённые подходы, которые позволили решить как проблемы автоматизации, так и проблемы последующего расширения системы непосредственно с помощью сценариев. А также получить ряд преимуществ перед альтернативными решениями. При этом особое внимание было уделено обеспечению дружественности программного интерфейса сценариев (Script API) для рядового пользователя, а также строгое следование принципам объектно-ориентированного программирования.

2. Обзор существующих решений

Большинство из ныне существующих систем оптического моделирования реализуют определенный уровень поддержки сценариев. В качестве примера таких систем можно привести Autodesk Maya, 3ds Max, CATIA, Rhinoceros 3D, и т.д. При детальном рассмотрении становится ясно, что эти системы используют похожие подходы для реализации работы со сценариями.

В первую очередь, все эти системы предоставляют возможности для запуска задач моделирования из командной строки. Благодаря этому становится возможной автоматизация с использованием текстовых файлов (пакетных файлов) содержащих последовательности требуемых команд. Множество всех возможных команд и их аргументов, как правило, детально описываются в документации конкретной системы. Сами команды, как правило, достаточно простые, поэтому их использование не требует какой-либо существенной квалификации от пользователя.

Также, в этих системах имеется специальный модуль, который называется интерпретатором сценариев. Его задача состоит в выполнении сценариев (скриптов) написанных на поддерживаемом системой языке, как правило, высокого уровня, например Python или Visual Basic. Интерпретатор сценариев производит трансляцию программных конструкций в соответствующие операции над данными, объектами и модулями системы оптического моделирования. Помимо интерпретатора сценариев, все системы также имеют специальный API (функциональный или объектно-ориентированный), который используется для непосредственного взаимодействия между скриптами и различными частями системы.

Ещё одним модулем, присутствующим во всех вышеперечисленных системах, является редактор сценариев. В большинстве случаев он представляет собой простейший текстовый редактор, с помощью которого можно создавать и редактировать сценарии, выполнять проверку их синтаксиса, отслеживать возможные ошибки в коде сценария. Часть систем поддерживает интеграцию с внешними редакторами, которые могут поддерживать дополнительные функции, такие как отладка сценариев во время выполнения, использование точек останова и т.д.

Рассмотрим более подробно особенности поддержки сценариев в некоторых вышеперечисленных системах.

Система оптического моделирования Autodesk Maya имеет поддержку сразу нескольких языков сценариев - MEL (Maya Embedded Language) и Python. Сценарии, написанные на языке MEL, используют процедурный MEL API [5], обладающий очень богатым функционалом, но являющийся крайне неудобным для пользователя из-за отсутствия объектной ориентации. Maya Python API [8, 9] свободен от этого недостатка, но этот API очень сложен для многих задач из-за своего весьма низкого уровня. Также следует отметить, что язык MEL использует собственный синтаксис, поэтому для работы с ним от пользователя требуются определенные навыки. Сценарии можно создавать и запускать непосредственно в Autodesk Maya с помощью встроенного редактора сценариев. Также поддерживается запуск сценариев в режиме командной строки. Сценарии можно сохранять в виде текстовых файлов в файловой системе, а также группировать в специальные библиотеки. Помимо всего этого, есть возможность создания плагинов на основе сценариев, расширяющих возможности системы.

Система оптического моделирования Rhinoceros 3D содержит специальный программный инструмент Rhinoscript, используемый для создания сценариев на основе языка высокого уровня Microsoft VBScript. Для взаимодействия сценариев и системы используется технология COM, посредством которой обеспечивается доступ из них к объектам системы. Сценарии можно запускать из встроенного редактора сценариев. Rhinoceros 3D имеет довольно удобный объектно-ориентированный Rhinoscript API. Этот API содержит более полутысячи различных

методов, позволяющих пользователю легко автоматизировать большинство прикладных задач [10]. В тоже время в реализации поддержки сценариев в Rhinoceros 3D есть некоторые недостатки. Во-первых, это невозможность создания новых типов и объектов или расширения существующих типов и объектов системы с помощью сценариев. Пользователь ограничен лишь теми типами и объектами, которые доступны через Rhinoscript API. Таким образом, добавление новой функциональности в систему Rhinoceros 3D возможно только разработчиками, а не конечными пользователями этой системы. Во-вторых, в Rhinoceros 3D отсутствует поддержка запуска сценариев из командной строки, что может затруднить автоматизацию части задач. Этот недостаток, на наш взгляд, является весьма существенным, учитывая тот факт, что сценарии нельзя размещать и сохранять непосредственно в сцене, следовательно, возможность их автоматического выполнения при загрузке сцены также не может быть реализована. Таким образом, работа со сценариями в системе Rhinoceros 3D всегда требует явного взаимодействия с пользователем, что усложняет задачи автоматизации.

Поддержка сценариев в САПР CATIA построена аналогичным образом с использованием технологии COM. В CATIA возможен запуск скриптов, как из графического интерфейса пользователя, так и из интерфейса командной строки. Одной из интересных особенностей 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 [11].

Рассмотрев существующие решения и подходы, мы реализовали собственный подход, устраняющий некоторые недостатки, имеющиеся в вышеперечисленных системах.

3. Объектно-ориентированный подход к поддержке сценариев на уровне ядра системы

Наша система оптического моделирования [1] представляет собой модульную систему, состоящую из различных компонентов, среди которых можно выделить две главные подгруппы – модули вычислительного ядра и модули графического интерфейса пользователя. Благодаря использованию строгой объектно-ориентированной структуры, весь функционал системы был сосредоточен во множестве компонентов вычислительного ядра, что позволяет использовать их в отрыве от других модулей системы. Эта возможность также делает возможным использование различных клиентских сторон пользовательского интерфейса (frontend), например графических, консольных, сетевых без существенной переделки всей системы оптического моделирования. Функциональная схема нашей системы показана на рисунке 1.

Для взаимодействия различных модулей между собой, нами используется специальный унифицированный целевой интерфейс объектов – `Entity Interface` [7]. Этот интерфейс встраивается в цепочку базовых классов всех компонентов нашей системы и предоставляет эффективный способ взаимодействия внутри неё с помощью специальных методов чтения/изменения свойств объектов, а также методов для выполнения различных действий над объектами. Им также поддерживается возможность создания новых объектов требуемого типа с конкретными параметрами. Подмножество классов унаследованных от базового класса, реализующего унифицированный целевой интерфейс объектов, формирует программный интерфейс (C++ API) нашей системы оптического моделирования.

Используя вышеперечисленные архитектурные решения нам удалось существенно упростить реализацию поддержки сценариев в системе.

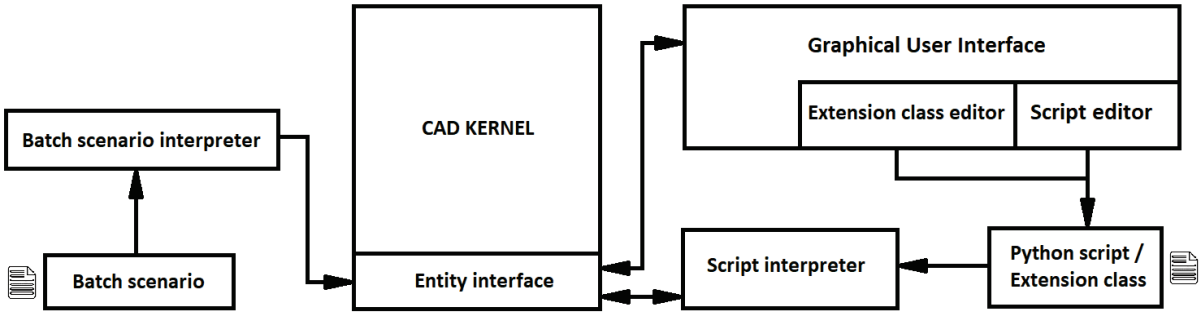

Рисунок 1 – Функциональная схема системы оптического моделирования

3.1. Интерпретатор пакетных сценариев

Благодаря модульной архитектуре, поддержка выполнения сценариев командной строки была реализована достаточно просто. Было создано специальное консольное приложение, слинкованное с ядром оптической системы. Это приложение используется для автоматизации наиболее часто использующихся задач моделирования, таких как:

- Загрузки различных моделей (сцен).
- Реалистичного рендеринга моделей методами Monte Carlo и т.д.
- Симуляции с использованием виртуальных измерительных приборов.
- Сохранения результатов моделирования.

Список доступных команд и их аргументов может быть запрошен непосредственно в самом приложении. Команды имеют простой синтаксис, и их использование требует от пользователя лишь базовых навыков работы с командной строкой. Ниже приведён пример команды позволяющей автоматизировать сразу несколько действий, а именно загрузку сцены, рендеринг модели и последующее сохранение результатов в файл:

```
Batch.exe Room_model.iof #rend -o room_model_out -i -l
```

3.2. Интерпретатор Python скриптов

Обратной стороной простоты пакетных сценариев, является отсутствие гибкости при решении комплексных задач автоматизации вычислений. Например, пользователю необходимо произвести изменение некоторых редко использующихся параметров модели, однако необходимая для этого команда отсутствует (не имплементирована) в пакетном интерпретаторе. При желании, поддержку отсутствующих команд можно добавить, однако это действие будет требовать определённых усилий со стороны разработчиков системы.

Этот и другие недостатки можно преодолеть, используя возможности скриптовых языков. Наш выбор пал на язык Python, из-за ряда его преимуществ, таких как простота, наличие элегантного дизайна и дисциплинирующего синтаксиса, широкой расширяемости, полной поддержки объектно-ориентированного программирования, облачных вычислений и средств масштабирования [2], доступности на различных платформах (Windows, Linux, 32/64 бит). Дополнительно стоит отметить наличие большого количества пакетов расширений для самых разных целей (numpy, scipy, matplotlib, imageio) [3, 4].

Поддержка языка сценариев Python в нашей системе оптического моделирования реализуется в виде отдельной библиотеки, содержащей интерпретатор этого языка, который взаимодействует с остальной системой, через унифицированный целевой интерфейс. Набор объектов, образующих C++ API нашего оптического комплекса естественным образом переносится на язык Python с помощью специального пакета CPython [6]. Получившийся в итоге интерфейс программирования сценариев (Python API) использует ту же самую иерархию объектов, что и комплекс оптического моделирования и, что самое главное, позволяет осуществлять доступ из сценариев к объектам системы, их свойствам и методам с помощью обычной точечной нотации. Такой подход позволяет разрабатывать скрипты с использованием

устоявшегося синтаксиса языка Python. В нашем случае нет необходимости использовать какие-либо дополнения или отдельные синтаксические конструкции для доступа к объектам системы из скриптов. От пользователя требуются лишь базовые знания языка Python.

Так как интерпретатор сценариев реализован в виде отдельного модуля, он может быть слинкован с ядром системы, как консольное приложение. В этом случае, для запуска сценария достаточно лишь передать этому приложению путь к файлу сценария. Если же предполагается использование графического интерфейса системы оптического моделирования, интерпретатор может взаимодействовать со специальным модулем пользовательского интерфейса - редактором сценариев. В этом случае, запуск сценариев выполняется непосредственно из редактора сценариев.

Модуль редактора сценариев является простым текстовым редактором, поддерживающим некоторые дополнительные функции, облегчающие разработку скриптов, такие как проверка ошибок, загрузка и сохранение скриптов в файловой системе, запуска сценариев. Кроме того, он поддерживает вывод результатов выполнения сценариев в специальное поле и предоставляет доступ к документации интерфейса программирования сценариев Python API. Также следует отметить, что в нашей реализации нет возможности автоматически генерировать сценарии (макросы) на основе действий пользователя в графическом интерфейсе, например, путем отслеживания событий мыши или клавиатуры.

На рисунке 2 показан интерфейс редактора сценариев. В качестве примера, в него загружен код скрипта выполняющий циклический рендеринг модели при различных значениях освещённости определяемой заданным временем суток.

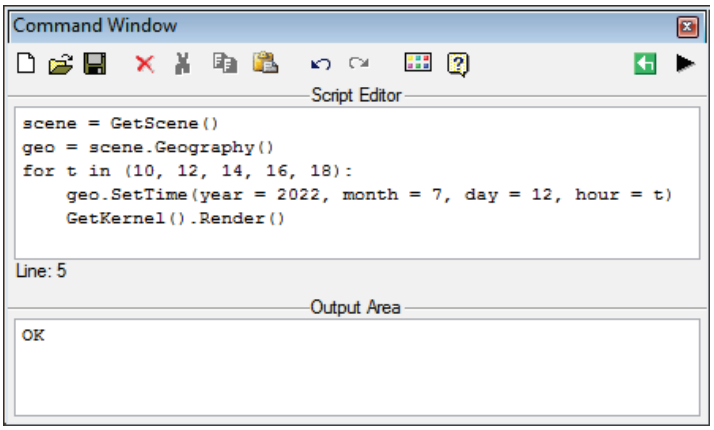

Рисунок 2 – Интерфейс редактора сценариев

3.3. Сценарные классы расширений

Вышеперечисленные подходы к использованию сценариев предполагают, что они хранятся вне самой модели, например, в виде отдельного файла на диске. Это приводит к тому, что изменение параметров модели или алгоритма моделирования требует изменения самого сценария. Чаще всего это может быть затруднено, так как на практике сценарии бывают большими и сложными. Нами был разработан особый подход, позволяющий обойти это ограничение.

Суть подхода заключается в использовании специальных параметрических объектов, которые с одной стороны являются частями модели, т.е. объектами сцены, а с другой стороны, содержат в себе скрипт на языке Python, представляющий из себя класс расширения.

В отличие от обычных Python скриптов, являющихся набором последовательно выполняющихся подпрограмм, скрипты классов расширений имеют более сложную структуру. Во-первых, они не имеют явной точки входа, с которой начинается их выполнение. Вместо этого, для взаимодействия с остальной системой они используют специальные функции обратного вызова, которые вызываются ядром комплекса оптического моделирования. Во вторых, классы расширений позволяют задавать, а затем многократно изменять параметры сценарного объекта, без изменения кода самого сценария. Пример простейшего класса

расширения, с помощью которого можно выполнить рендеринг модели при различных значениях облачности, показан на рисунке 3

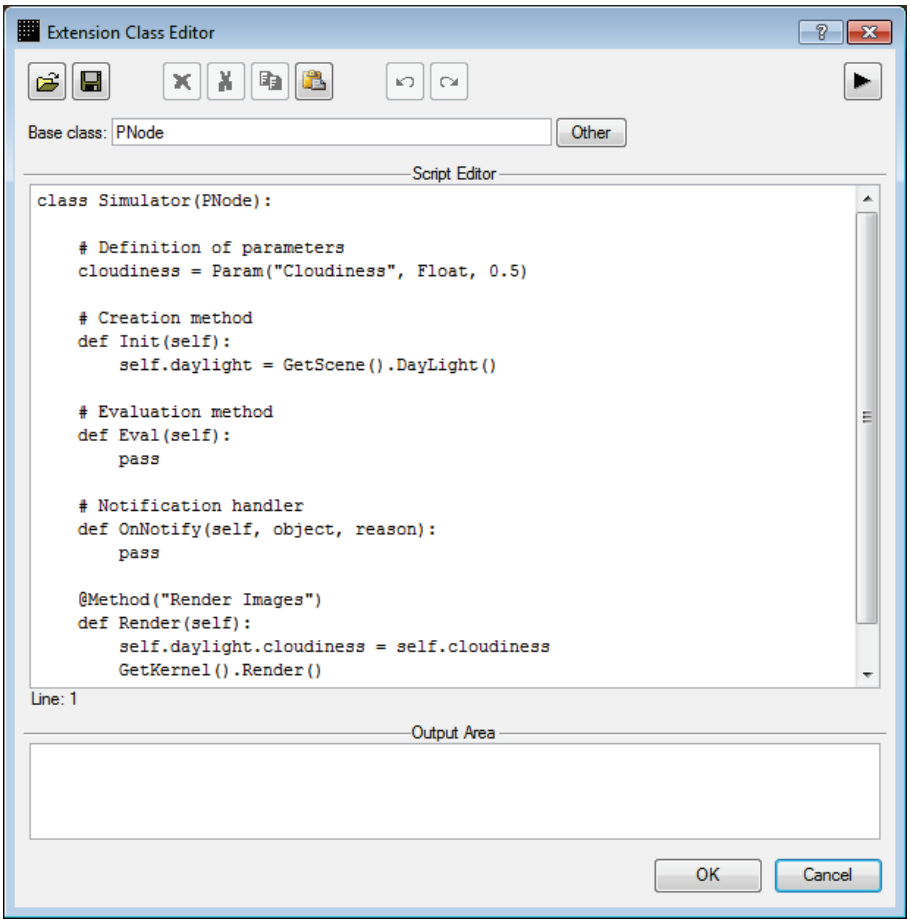

Рисунок 3 – Скриптовый класс расширения, загруженный в редактор

Как видно из примера, классы расширений имеют достаточно простую структуру, состоящую из заголовка определяющего имя класса и последующих определений параметров и методов класса. Параметрами класса расширения могут быть как простые типы данных, например скаляры или строки, так и сложные типы, такие как массивы и таблицы. Для определения параметров используется специальный метод `Param()`. В нашем примере мы определяем единственный параметр – облачность. Дальнейшее изменение параметров возможно как из самого сценария, так и из графического интерфейса пользователя.

Другими специальными методами, использующимися всеми без исключения классами расширений, являются методы `Init()` и `Eval()`. Метод `Init()` является конструктором и выполняется только один раз при создании экземпляра объекта класса расширения. Метод `Eval()` автоматически вызывается ядром комплекса оптического моделирования при изменении параметров класса. Целью этого метода является обновление представления объекта в соответствии с новыми значениями параметров.

В отличие от прочих типов сценариев, классы расширения могут взаимодействовать (встраиваться) с графическим интерфейсом пользователя. На данный момент, была реализована лишь базовая поддержка данной функциональности, позволяющая включить любой из методов скриптового класса расширения в контекстное меню параметрического объекта, как это показано на рисунке 4. Для этого перед требуемым методом используется специальный декоратор `@Method()`. Учитывая, что объекты классов расширений являются частью модели, доступ к ним, а также к коду сценария может осуществляться с помощью графического интерфейса пользователя.

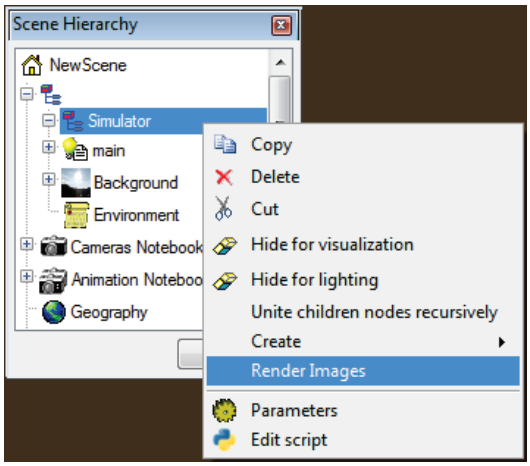

Рисунок 4 – Метод класса расширения «Render Images» доступный из меню параметрического объекта

3.4. Механизм уведомлений в классах расширений

Одним из недостатков поддержки сценариев в рассмотренных выше системах оптического моделирования является отсутствие механизма уведомлений об изменении состояния тех или иных объектов модели. Этот недостаток может накладывать существенные ограничения на задачи автоматизации моделирования и требовать от пользователя использования нетривиальных решений, существенно усложняющих сценарии.

В нашей системе данный недостаток был устранён. Для этого, в сценариях был реализован механизм уведомлений, основывающийся на низкоуровневом механизме уведомлений, который предоставляется унифицированным целевым интерфейсом объектов (`Entity Interface`) и является его неотъемлемой частью. Функционал классов расширений был дополнен специальной функцией обратного вызова `OnNotify()` являющейся обработчиком уведомлений и получающей в качестве аргументов как указатель на объект пославший уведомление, так и идентификатор (тип) уведомления. Дополнительно, в классах расширений была реализована возможность подписываться и отписываться на/от уведомления от различных объектов системы. Важно отметить, что наша реализация поддерживает работу с уведомлениями, как от обычных, так и от сценарных объектов системы. Ниже приведён пример класса расширения, использующий механизм уведомлений. В данном примере, уведомления используются для автоматического изменения положения геометрии в пространстве вслед за изменением положения интересуемого источника освещения.

```
class WatchingNode(PNode):

    # Creation method
    def Init(self):
        # Enable receiving notifications from interesting node
        self.EnableNotify(GetScene().GetNode("Light node"))

    # Evaluation method
    def Eval(self):
        pass

    # Notification handler
    def OnNotify(self, obj, reason):
        # Check the object that issued the notification
        s = GetScene()
        if ((s != None) and (obj == s.GetNode("Light node"))):
            # Check reason
```

```
if ((reason is not None) and (reason ==
NodeNotifyReason.TRANSFORM_CHANGED)):
    # Point light node transform was changed, change
    Box shape node transform accordingly
    s.GetNode("Box").tr.rot_matr4 = obj.tr.rot_matr4
```

4. Заключение

Описанные подходы по использованию различных типов сценариев при решении задач автоматизации вычислений в настоящее время широко используются в разработанном нами программном комплексе оптического моделирования. Отдельно надо отметить, что сценарии и классы расширений используются не только пользователями системы, но и самими разработчиками, например, для упрощения задач жизненного цикла разработки приложений, таких как автоматическое тестирование и диагностика.

Совершенствование сценарной поддержки продолжается и в текущий момент. Главным направлением развития является улучшение возможностей скриптовых классов расширений, в плане повышения интерактивности основанных на них параметрических объектов, и предоставления ими более удобных методов для эффективной реализации разнообразных задач автоматизации моделирования.

5. Список источников

[1] Интеграция реалистичной графики в системы автоматизированного проектирования и управления жизненным циклом изделия / Б.Х. Барладян, А.Г. Волобой, В.А. Галактионов, Л.З. Шапиро. // Программирование, 2018, № 4, с. 26-35.

[2] Ami Marowka. Python accelerators for high-performance computing. // The Journal of Supercomputing, 74.4, 2018, pp. 1449-1460.

[3] NumPy The fundamental package for scientific computing with Python [Электронный ресурс]. URL: https://numpy.org (дата обращения 19.05.2022).

[4] scikit-image: image processing in Python / Stefan Van der Walt, et al // PeerJ 2 e453 (2014).

[5] PyMEL for Maya [Электронный ресурс]. URL: https://help.autodesk.com/cloudhelp/2018/JPN/Maya-Tech-Docs/PyMel/index.html (дата обращения 19.05.2022).

[6] N.B. Deryabin, D.D. Zhdanov, V.G. Sokolov. Embedding the Script Language into Optical Simulation Software // Programming and Computer Software, 2017, Vol. 43, №. 1, pp. 13-23.

[7] В.А. Галактионов, Н.Б. Дерябин, Е.Ю. Денисов. Объектно-ориентированный подход к реализации систем компьютерной графики // «Информационно-измерительные и управляющие системы», № 6, 2009, с. 96-108.

[8] Gilenn Emille G. Collado. Modeling and Python Scripting in Maya for the Animation Short Style // Department of Computer Engineering California Polytechnic State University, San Luis Obispo, CA, 2016.

[9] Adam Mechtley, Ryan Trowbridge. Maya Python for Games and Film: A Complete Reference for the Maya Python API. CRC Press, 2011.

[10] Lagios Kera, Jeff Niemasz, Christoph F. Reinhart, Animated building performance simulation (ABPS)–linking Rhinoceros/Grasshopper with Radiance/Daysim, Conference proceedings of 4-th National Conference of IBPSA-USA, August 11-13, 2010, New York City, New York, pp. 321-327.

[11] Yihui Li. Research of Integration Technology between CATIA and TOOLMANAGER Based on CAA, International Journal of Advanced Network, Monitoring and Controls 1.1 (2018).