# Особенности использования сценариев в автоматическом тестировании систем оптического моделирования

Копылов М.С. [1], Бирюков Е.Д. [1], Барладян Б.Х. [1]
kopylov@gin.keldysh.ru|birukov@gin.keldysh.ru|obb@gin.keldysh.ru
[1]ИПМ им. М.В. Келдыша РАН, Москва, Россия

*В работе рассмотрен опыт применения сценариев, написанных на языках Python и Visual Basic, для автоматического тестирования отдельных программных компонентов комплекса оптического моделирования. Также рассмотрены аспекты отдельного тестирования пользовательского интерфейса такого комплекса. Приведены примеры подобного тестирования.*

***Ключевые слова**: автоматическое тестирование, сценарии, графический интерфейс.*

# Usage of scenarios for automatic testing of optics CAD systems

Kopylov M.S. [1], Birukov E.D. [1], Barladian B.Kh. [1]
kopylov@gin.keldysh.ru|birukov@gin.keldysh.ru|obb@gin.keldysh.ru
[1] Keldysh Institute of Applied Mathematics, Moscow, Russia

*This work describes specific features of Python and Visual Basic scenarios usage for automatic testing of individual program components of the 3D optical simulation software package. Also there are described aspects of a special testing of graphical user interface for such package. Examples of such testing are provided.*

***Keywords**: automatic testing, scripts, graphical interface.*

## 1. Введение

Увеличение сложности программного обеспечения требует максимального использования всех доступных средств автоматизации разработки. Тестирование программного обеспечения, как одна из дорогостоящих процедур, нуждается в автоматизации на всех этапах жизненного цикла. Это особенно критично в условиях ограниченных ресурсов и времени на выпуск новых версий продукта.

Для программных комплексов, выполняющих расчеты различных физических процессов, к которым относятся системы оптического моделирования, наиболее важным видом тестирования является тестирование корректности результатов, которое может выполняться путем сравнения с аналитическими результатами или результатами полученными другими независимыми программами.

Регрессионное тестирование основано на сравнении полученных результатов в текущей версии программы с аналогичными результатами, полученными в предыдущей версии программы, которые считаются заведомо правильными.

Кроме того, в системах автоматизированного проектирования требуется тестирование корректности работы пользовательского интерфейса, зачастую имеющего сложную разветвленную структуру.

## 2. Использование сценариев при тестировании компонентов ядра комплекса

Комплекс оптического моделирования Lumicept, разрабатываемый нашим коллективом в ИПМ им. М. В. Келдыша РАН представляет собой модульную систему, состоящую из различных компонентов, которые можно разделить на две подгруппы – модули ядра и модули графического интерфейса. При этом множество компонентов ядра содержит в себе почти весь функционал комплекса, поэтому задача тестирования этой функциональности является первоочередной.

Комплекс Lumicept также интегрирован с существующей системой автоматизированного проектирования CATIA, которая является одной из популярных САПР в области машиностроения, в частности, проектирования автомобилей и самолетов. Интеграция выполнена в виде специальных дополнений (add-ons) к системе CATIA, которые вызывают общие модули ядра (в некоторых случаях – и общие модули графического интерфейса) [1].

Основные модули комплекса Lumicept имеют встроенную систему обработки сценариев на базе языка Python. Подмножество внутренних объектов и классов данного комплекса доступно из этих сценариев через Python API [2]. Данная особенность даёт возможность автоматизировать действия пользователя по работе с различными объектами данного комплекса, такими как объекты сцены, модули рендеринга, различные симуляторы и.т.д. К настоящему времени общее число подобных объектов весьма велико, например одних только объектов сцены существует более сотни. Так же необходимо отметить, что многие из этих объектов имеют достаточно большое количество различных настроек, влияющих на режимы их работы и вычислений.

Сценарии можно запускать как в режиме командной строки, так и в режиме графического интерфейса. Как правило, при автоматическом тестировании используется режим командной строки. Запуск сценариев осуществляется с помощью консольной программы ipython.exe, которая представляет собой специальный модуль комплекса Lumicept, содержащий в себе интерпретатор языка Python с возможностью доступа ко всем остальным объектам комплекса. Пример простейшего скрипта на языке Python приведен ниже.

```
def Render(file):
    kernel = Kernel()
    kernel.LoadScene(file)
    rp = RenderParams()
    rp.res = (800, 600)
    rp.depth = 5
    res = kernel.Render(rp)
    res.SaveImageToFile(…)
    res.CompareImageWithEthalon(…)

files = [“file1”, “file2”, …, “file99”]
```


The publication of the conference proceedings was supported by the RFBR, grant No. 18-07-20045\18

```
for f in files:
    Render(f)
```

В данном примере в ядро комплекса Lumicept последовательно загружаются файлы, содержащие в себе различные модели (сцены). Затем производится рендеринг каждой модели и результат (как правило, это графическое изображение) сохраняется в новый файл. На заключительном шаге производится сравнение получившегося изображения с эталонным изображением, с помощью специальной подпрограммы.

Работа со сценариями состоит из нескольких обязательных этапов - подготовка исходных данных, выполнение расчётов, сохранение и анализ результатов. В качестве необязательного этапа можно добавить установку или настройку различных параметров модулей, которые задействованы в вычислениях. Сценарии не имеют каких-либо ограничений по длине (количеству операторов), а также могут вызвать другие сценарии, быть вложенными, использовать рекурсию и.т.д. При этом из любого сценария можно получить доступ как к одному, так и к нескольким модулям комплекса оптического моделирования одновременно.

Подобная широкая функциональность, несомненно, является весьма удобной для автоматизации вычислений. В рамках тестирования же, как правило, ограничиваются работой только с одним конкретным модулем в каждый момент времени, то есть для каждого модуля или компонента модуля пишется отдельный тестовый сценарий, который используется для тестирования работоспособности этого компонента или модуля. Укрупнение подобных сценариев тоже возможно, например, объединив их в пакет подпрограмм в рамках одного сценария.

Особенностью реализации Python API, разрабатываемой нами для написания тестовых сценариев, является строгая проверка на ошибки, возникающие по ходу выполнения сценариев. Допустим, если по каким либо причинам исходный файл не сможет загрузиться, например, из-за ошибки в тестируемом модуле, то Python API немедленно вернёт код ошибки и выполнение сценария будет остановлено, что в свою очередь будет явно сигнализировать о возможных проблемах в данном модуле. Важно отметить, что даже успешное завершение тестового сценария не гарантирует правильности работы тестируемого модуля. Поэтому любой тестовый сценарий должен дополнительно выполнять анализ получившихся результатов. Обычно такой анализ представляет собой сравнение различных численных и графических данных, полученных при оптических симуляциях в загруженной сцене, например, результатов рендеринга.

Дополнительно требуется анализ скорости работы тестового сценария и количества используемой памяти в процессе работы. Выбор того или иного типа анализа зависит непосредственно от формата исходных данных, а также от теста (функциональный или регрессионный). Учитывая богатые возможности языка сценариев Python в плане обработки данных и в наличии множества дополнительных пакетов расширения самого разного назначения, таких как numpy, scipy, matplotlib, imageio, задачи анализа получившихся результатов могут быть реализованы непосредственно в сценарии тестирования, обычно в виде отдельной подпрограммы. Например, если результатами некоторых симуляций являются графические изображения, то их можно сравнивать с помощью библиотеки scikit-image (часть пакета scipy) для поиска различающихся регионов в изображениях [7]. В некоторых случаях (например, при проведении трассировки лучей методом Path Tracing) в вычислениях используются случайные величины, поэтому результаты всегда будут немного отличаться от запуска к запуску. В этих случаях для успешного прохождения теста среднеквадратичное отклонение между сравниваемыми изображениями не должно превышать некоторого определённого значения.

## 3. Многопоточность в сценариях

При автоматическом тестировании компонентов комплекса оптического моделирования весьма важным критерием является эффективность данной процедуры с точки зрения затраченного времени. К примеру, в настоящий момент, для выполнения всех тестов использующих сценарии может понадобиться от 12 до 15 часов (на компьютере со следующими характеристиками: Процессор: Intel Core i7-3770 3,4 ГГц; RAM: 32 Гб; ОС: Windows 8.1 x64). Поэтому в рамках работы с комплексом были выявлены задачи тестирования, выполняющиеся не оптимально с точки зрения использования доступных вычислительных ресурсов, характеризующиеся неполной загрузкой всех доступных процессоров или процессорных ядер. В основном это вызвано такими причинами, как собственные ограничения алгоритмов обработки графических данных в многопроцессорных/многоядерных конфигурациях и конфигурациях с неоднородным доступом к памяти (NUMA), или же из-за использования тестовыми сценариями некоторых специфических методов ядра комплекса, которые являются однопоточными по своей природе.

К счастью, язык сценариев Python обладает достаточно продвинутой поддержкой многопоточного программирования, позволяющей весьма сильно увеличить эффективность выполнения вышеперечисленных групп сценариев. Данным языком, например, поддерживаются оба базовых вида распределения вычислительной нагрузки - многопоточность и многопроцессность, которые реализованы в модулях threading и multiprocessing соответственно. При этом стоит отметить, что унификация интерфейсов данных модулей делает их весьма удобными в использовании. В некоторых случаях в исходном сценарии достаточно изменить лишь несколько строк кода, чтобы перейти от использования одного модуля к другому и наоборот. Более того, оба модуля поддерживают один и тот же набор примитивов синхронизации, что упрощает разработку программ. Выбор того или иного способа реализации многопоточности в тестовом сценарии, как правило, зависит от конкретных задач, решаемых им, и выбирается разработчиком данного сценария.

Как показывает наш опыт, модуль threading имеет некоторые ограничения при работе с потоками в плане эффективности, если эти потоки в основном работают с общими данными, такими как массивы, словари, списки. Другое ограничение связано с возможностью ситуации, при которой из двух потоков одновременно будет вызвана одна и та же функция ядра комплекса. Учитывая, что многие функции ядра комплекса не являются потокобезопасными, существует реальная угроза аварийного завершения при выполнении многопоточных тестовых сценариев, или возникновения ошибок в результатах вычислений или тестирования. В связи с этим, использование модуля multiprocessing является более предпочтительным при использовании в тестовых сценариях, так как при использовании процессов можно добиться большего параллелизма, особенно при задачах пакетной обработки данных, при этом данный модуль не имеет каких либо ограничений [6].

Пример многопоточного тестового сценария, накладывающий фильтр на множество отдельных изображений, приведен ниже. В данном примере каждый


Работа опубликована при финансовой поддержке РФФИ, грант 18-07-20045\18

экземпляр подпрограммы наложения фильтра на исходные изображения, выполняется в отдельном процессе, благодаря использованию библиотеки multiprocessing позволяющей вынести код любой функции сценария в отдельный процесс.

```
def ApplyFilter(name):
    pp = PostProcessor(name)
    res = pp.ApplyAverageFilter()
    res.SaveImageToFile(…)
    res.CompareImageWithEthalon(…)

if __name__ == '__main__':
    from multiprocessing import Process
    files = ["file1", "file2", … ]
    for f in files:
        p = Process(target = ApplyFilter, args = f)
        p.Start()
```

## 4. Особенности работы сценариев в САПР CATIA

Существует реализация программы Lumicept, которая представляет собой дополнения к САПР CATIA. Эта САПР использует технологию COM для работы встроенной системы сценариев. Данная технология предполагает, что исходный код всех объектов, которые существуют в самой системе CATIA, а также те, которые были добавлены к CATIA в различных дополнениях, должен быть написан по определенному стандарту. При этом та часть исходного кода для каждого объекта, которая содержит объявление этого объекта со списком находящихся в нем свойств и методов, является открытой. Эта часть написана на специальном языке IDL. Все объекты, написанные по стандартам COM, доступны с помощью пользовательских сценариев, при этом теоретически возможно использование любого языка сценариев, реализация которого поддерживает стандарт COM. Имена для объектов, их свойств и методов при написании сценариев берутся из исходного кода на языке IDL, приложенному к системе CATIA и к дополнениям к ней.

Существуют три способа вызвать запуск выполнения сценария в системе CATIA. Первые два способа предполагают использование команд самой системы CATIA для запуска сценария – непосредственно с помощью команды «Запуск сценария» из пользовательского интерфейса, либо при запуске CATIA из командной строки в качестве дополнительного параметра. Третий способ предусматривает вызов внешнего интерпретатора сценариев, который уже самостоятельно запускает систему CATIA и обеспечивает передачу обращений к объектам системы CATIA с помощью технологии COM. Этот последний способ позволяет использовать любой язык сценариев, поддерживающий стандарт COM, но, к сожалению, позволяет запустить систему CATIA только в консольном режиме, без отображения пользовательского интерфейса, поэтому не может использоваться для тестирования самого пользовательского интерфейса. Первые два способа могут использоваться как при консольном, так и при графическом режиме отображения системы CATIA, однако, в этом случае доступны только те языки сценариев, которые встроены в саму систему CATIA. В настоящее время такие языки представлены только несколькими разновидностями языка Basic.

Поскольку для программы Lumicept стандартом сценариев является Python, то этот язык применяется и при тестировании дополнения Lumicept для системы CATIA в тех случаях, когда это возможно. Однако в тех случаях, когда использование языка Python не представляется возможным (в первую очередь – при тестировании графического интерфейса модулей Lumicept в системе CATIA) – тогда используется язык CATVBs.

## 5. Методы тестирования пользовательского интерфейса САПР

Автоматическое тестирование пользовательского интерфейса, как правило, проводится с помощью отдельной программы. Для тестирования комплекса оптического моделирования Lumicept используется инструмент AutoIt, который имеет собственный встроенный язык сценариев, похожий на Basic [3]. Для имитации действий пользователя эти сценарии используют систему идентификаторов элементов управления в интерфейсе, и команд, которые можно посылать данным элементам управления. В операционных системах семейства Windows используются дескрипторы окон (при этом под окнами понимаются любые элементы управления), и сообщения, которые можно послать окнам с соответствующими дескрипторами с помощью функции SendMessage [4].

Если каждый элемент управления однозначно реагирует на действия мыши и клавиатуры, при этом достаточно лишь чтобы указатель мыши находился над любой точкой элемента управления, то такая система автоматического тестирования обеспечивает полную эмуляцию действий пользователя. Однако отличительной особенностью систем автоматизированного проектирования является необходимость выделять с помощью мыши не только стандартные оконные элементы управления, но также различные объекты, отображенные в основном окне просмотра. С точки зрения операционной системы окно отображения сцены является одним элементом управления, и объекты, нарисованные в этом окне, не имеют никаких внешних идентификаторов, по которым их можно найти сторонней программой. В большинстве систем автоматизированного проектирования существует то или иное отображение иерархической структуры объектов в сцене в виде дерева, обычно отображаемое в отдельном окне. Зачастую такое окно является стандартным элементом управления (например, TreeViewControl в операционной системе Windows), что позволяет выделять объекты в сцене с помощью такого дерева. Однако система автоматизированного проектирования CATIA отличается от других подобных систем тем, что дерево объектов сцены отображается в том же окне, что и сама сцена [5]. Стандартное окно системы CATIA с отображенным в нем деревом сцены показано на рис. 1.

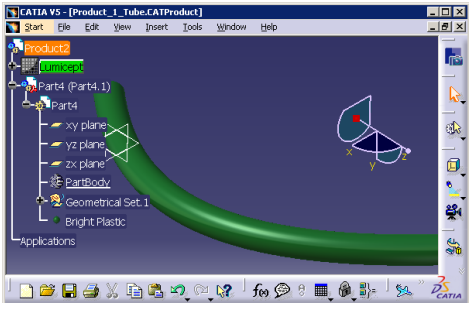


**Рис. 1.** Окно системы CATIA с загруженной сценой и отображенным деревом иерархии сцены

Ни само дерево, ни его отдельные элементы не имеют внешних идентификаторов. Поэтому в любом случае необходимы некоторые дополнительные действия для


The publication of the conference proceedings was supported by the RFBR, grant No. 18-07-20045\18

выделения в окне просмотра некоторой точки, соответствующей заданному объекту. Автоматизация тестирования пользовательского интерфейса в этом случае может быть произведена несколькими способами.

Первый способ выделения мышью определенной точки на элементе интерфейса предусматривает жесткое прописывание в сценарии теста конкретных координат этой точки. Второй способ заключается в использовании систем распознавания изображений. Оба этих способа имеют свои недостатки: в первом случае – необходимость использования строго определенного разрешения экрана и масштаба элементов управления, во втором случае – общую неточность работы систем распознавания изображений, особенно, если на экране присутствуют несколько похожих друг на друга объектов.

Третий способ, выбранный авторами в качестве основного, представляет собой комбинацию использования сценариев, встроенных в тестируемую программу, и внешней программы автоматических тестов пользовательского интерфейса. Основу тестов составляет сценарий, выполняемый в самой тестируемой программе. Этот сценарий вызывает внешнюю программу тестирования пользовательского интерфейса (в нашем случае – AutoIt) непосредственно в тех местах, где требуется тестирование отдельных элементов интерфейса.

Рассмотрим для примера тестирование диалогового окна редактирования параметров материала в системе CATIA. Данное диалоговое окно является частью самой системы CATIA, однако в рамках комплекса оптического моделирования, разработанного нашим коллективом, к нему были добавлены дополнительные параметры. При тестировании используется сценарий на языке CATVBs (разновидность языка Visual Basic), который вызывается из самой системы CATIA, а также внешний инструмент AutoIt и его встроенный язык сценариев.

Тестирование состоит из следующих этапов:

1. Запуск системы CATIA в стандартном режиме с графической оболочкой, но с автоматическим запуском сценария. Для запуска используется пакетный командный файл (BAT), в котором заданы параметры – необходимая оболочка (Workbench) системы CATIA (в данном случае – оболочка Lumicept), а также путь к сценарию, который будет запущен сразу после запуска. Следует обратить внимание на то, что в данном случае система CATIA будет открыта в стандартном графическом режиме, хотя, как правило, при автоматическом запуске сценариев используется консольный режим.

2. Сценарий (назовем его сценарий 1) на языке CATVBs, который выполняет действия над объектами сцены. Данный сценарий производит поиск материала с заданным именем, добавляет его в список выделенных объектов, вызывает стандартную команду Properties(), которая открывает диалог параметров выделенного объекта, после чего вызывает программу AutoIt для автоматического тестирования диалога параметров. Для вызова AutoIt используется команда SystemService.ExecuteProcessus() из набора системных процедур CATIA. Эта команда ждет завершения работы программы AutoIt; на протяжении этого времени диалог параметров остается открытым. Его закрытие произойдет уже далее, с помощью сценария AutoIt, который будет имитировать нажатие пользователем кнопки OK. После закрытия диалога параметров управление снова переходит к сценарию 1, который считывает значения требуемых параметров из объекта материала и проверяет, что они были правильно заданы с помощью диалога параметров.

3. Сценарий (назовем его сценарий 2) для работы инструмента AutoIt. Этот сценарий находит диалоговое окно с заголовком "Properties", проверяет, что это окно принадлежит системе CATIA с помощью сравнения идентификаторов главного окна системы CATIA и данного диалогового окна. Затем AutoIt производит активацию этого окна, после чего передает фокус ввода в элемент управления (поле ввода) с именем "SpnDifReflPos", записывает в это поле текст "0.5" и инициирует нажатие кнопки OK. Управление снова передается самой системе CATIA, в которой продолжает работу сценарий 1. Он проверяет правильность установленных данных.

## 6. Заключение

Система автоматических тестов на базе сценариев, описанная в данной статье, в настоящее время используется при разработке системы оптического моделирования и реалистичной компьютерной графики, которая ведется в ИПМ им. М.В. Келдыша РАН. Также начала внедряться система тестирования пользовательского интерфейса того же программного комплекса. При этом тестированию, в том числе, стало возможным подвергать ту часть комплекса, которая представляет собой дополнения к системе автоматизированного проектирования CATIA.

## 7. Литература

[1] Барладян Б.Х., Дерябин Н.Б., Шапиро Л.З. Интеграция модуля компьютерной графики в систему CATIA // Новые информационные технологии в автоматизированных системах: материалы шестнадцатого научно-практического семинара - Моск. Ин-т электроники и математики национального исследовательского университета «Высшая школа экономики». М., 2013, с. 11-19.

[2] Н.Б. Дерябин, Д.Д. Жданов, В.Г. Соколов. Внедрение языка сценариев в программные комплексы оптического моделирования // Программирование, 2017, № 1, с. 40-53.

[3] Денисов Е.Ю., Волобой А.Г., Калугина И.А. Автоматическое тестирование графического интерфейса интерактивных программных комплексов // Новые информационные технологии в автоматизированных системах: материалы двадцатого первого научно-практического семинара - М.: ИПМ им. М.В.Келдыша, 20 апреля 2018, с. 83-88.

[4] Atif Memon, Adithya Nagarajan, 2003. GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing. In Proceedings of the 10th Working Conference on Reverse Engineering (WCRE '03)

[5] Dassault Systemes, Inc. CATIA Vertsion 5-6 R2015 Documentation, 2014

[6] Singh, Navtej, Lisa-Marie Browne, and Ray Butler. 2013. "Parallel astronomical data processing with Python: Recipes for multicore machines." Astronomy and Computing 2: 1-10.

[7] Van der Walt S, Schönberger JL, Nunez-Iglesias J, Boulogne F, Warner JD, Yager N, Gouillart E, Yu T. scikit-image: image processing in Python. PeerJ. 2014 Jun 19;2:e453.

## Об авторах

Михаил Сергеевич Копылов, ст. инженер, ИПМ им. М.В. Келдыша РАН. E-mail: pmk@gin.keldysh.ru.

Елисей Дмитриевич Бирюков, м.н.с., ИПМ им. М.В. Келдыша РАН. E-mail: peb@gin.keldysh.ru.

Борис Хаимович Барладян, к.т.н., с.н.с., ИПМ им. М.В. Келдыша РАН. E-mail: obb@gin.keldysh.ru.



Работа опубликована при финансовой поддержке РФФИ, грант 18-07-20045\18