DOI: 10.15514/ISPRAS-2020-32(1)-4

Технологии автоматического тестирования программных комплексов реалистичной компьютерной графики

Е.Ю. Денисов, ORCID: 0000-0002-0614-9100 <eed@spp.keldysh.ru>
А.Г. Волобой, ORCID: 0000-0003-1252-8294 <voloboy@gin.keldysh.ru>
Е.Д. Бирюков, ORCID: 0000-0003-4297-6813 <birukov@gin.keldysh.ru>
М.С. Копылов, ORCID: 0000-0002-9526-0766 <pmk@gin.keldysh.ru>
И.А. Калугина, ORCID: 0000-0001-5558-3045 <qik@gin.keldysh.ru>
Институт прикладной математики им. М.В.Келдыша РАН,
125047, Россия, Москва, Миусская пл., д.4

Аннотация. В статье описаны технологии автоматического тестирования программного обеспечения применительно к промышленным системам компьютерной графики и оптического моделирования. Автоматизация тестирования становится жизненно необходимой в условиях ограниченности ресурсов при частом выпуске версий, которые нередко возникают у производителей программного продукта. Представлены как методы регрессионного тестирования вычислительного ядра таких комплексов, так и способы тестирования пользовательского интерфейса. Для регрессионного тестирования используется механизм сценариев на языке Python. Рассмотрены методы его распараллеливания, которые позволяют значительно сократить время тестирования. Поскольку в оптическом моделировании широко применяются стохастические методы, результаты расчетов могут отличаться, что осложняет регрессионное тестирование. В этом случае предлагается применять некоторый порог при сравнении результатов. Автоматизированные тесты для тестирования пользовательского интерфейса разработаны на основе инструмента AutoIt. Отдельно описаны подходы к тестированию пользовательского интерфейса систем, реализованных в виде дополнений (plug-in) к существующим комплексам автоматизации проектирования, исходный код которых закрыт и недоступен для авторов автоматических тестов.

Ключевые слова: тестирование ПО; автоматическое тестирование; компьютерная графика; моделирование освещенности; надежность программного продукта.

Для цитирования: Денисов Е.Ю., Волобой А.Г., Бирюков Е.Д., Копылов М.С., Калугина И.А. Технология автоматического тестирования программного комплекса реалистичной компьютерной графики. Труды ИСП РАН, том 32, вып. 1, 2020 г., стр. 71–88. DOI: 10.15514/ISPRAS–2020–32(1)–4

Technologies for automatic testing of a software package for realistic computer graphics

E.Y. Denisov, ORCID: 0000-0002-0614-9100 <eed@spp.keldysh.ru>
A.G. Voloboy, ORCID: 0000-0003-1252-8294 <voloboy@gin.keldysh.ru>
E.D. Birukov, ORCID: 0000-0003-4297-6813 <birukov@gin.keldysh.ru>
M.S. Kopylov, ORCID: 0000-0002-9526-0766 <pmk@gin.keldysh.ru>
I.A. Kalugina, ORCID: 0000-0001-5558-3045 <qik@gin.keldysh.ru>
Keldysh Institute of Applied Mathematics of RAS,
125047, Russia, Moscow, Miusskaya sq., 4

Abstract. The article describes the technology of automatic software testing in relation to industrial systems of computer graphics and optical simulation. Test automation becomes vital in the face of limited resources with the frequent release of product versions, which often occur among software product manufacturers. There are presented both methods of regression testing the computational kernel of such systems, and methods of testing the user interface. Scripting mechanism based on Python is used for regression testing, its multithreading capabilities which allow significant decreasing of testing time are also described. Python allows two ways of parallelization – multithreading and multiprocessing, both of them are considered. Due to the stochastic methods used in optical simulation calculation results may differ from time to time, which complicates regression testing. In this case, it is proposed to apply some (in each case - your own) threshold when comparing the simulation results. Separately automated testing of user interface which was elaborated basing on the AutoIt tool is described. The approach for testing the user interface of systems implemented in the form of plugins to existing CAD/PDM complexes, the source code of which is closed and not available to the authors of automatic tests, are described as well.

Keywords: software testing; automatic testing; computer graphics; lighting simulation; software product robustness.

For citation: Denisov E.Y., Voloboy A.G., Birukov E.D., Kopylov M.S., Kalugina I.A. Technology for automatic testing of a software package for realistic computer graphics. Труды ИСП РАН, том 32, вып. 1, 2020 г., стр. 71–88. DOI: 10.15514/ISPRAS–2020–32(1)–4

1. Введение

Создание современных программных комплексов невозможно без автоматизации процесса их разработки и сопровождения [1]. Одним из важнейших этапов жизненного цикла программного продукта является тестирование. Полноценное тестирование требует проверки различных аспектов программы, таких как наличие требуемой функциональности, быстродействие имеющихся алгоритмов, их корректную работу при различных исходных данных, а также пользовательский интерфейс и его корректную реакцию на различные действия пользователя, в том числе ошибочные. Сложность современных программных комплексов реалистичной графики и оптического моделирования такова, что классическое тестирование [2] становится просто неэффективным. Оценки ресурсов и времени, необходимых для полного тестирования таких комплексов, очень высоки. Особенностями тестирования таких комплексов можно назвать использование общих компонент разнородными программными компонентами и комплексами. При необходимости выпускать частые новые версии ПО в жестких ограничениях на сроки их выпуска автоматизация процесса тестирования становится неизбежной

В последнее время широко распространяются системы автоматического тестирования программного обеспечения. Такие системы позволяют существенно увеличить объем тестирования при одновременном снижении усилий [3]. Особенно эффективно использование автоматического тестирования при большом количестве однотипных тестов, например, при тестировании работы одного алгоритма с разными наборами

исходных данных или тестировании поведения нескольких однотипных диалоговых окон [4]. В основном используется два типа тестирования: тестирование на основе аналитических расчётов, обеспечивающее сравнение результатов работы системы с ожидаемыми, и так называемое регрессионное тестирование, обеспечивающее совпадение поведения новой реализации системы и её предыдущей реализации.

Наш опыт показывает, что иногда ошибки, однажды уже исправленные в программе, могут опять возникнуть в будущих версиях программного продукта. Такие ситуации нечасты, но и «единичным случаем» их назвать нельзя. Надо заметить, что причиной повторного появления уже исправленных ошибок могут быть как ошибки интеграции исходных кодов, когда необходимая ревизия кода была просто утеряна или пропущена, так и совершенно новые независимые ошибки, которые приводят к такому же внешнему проявлению, что и ранее исправленная ошибка. Но для конечного пользователя программы, которому неизвестны истинные причины повторного появления ошибки, эта причина не важна. Факт повторного проявления уже однажды найденной и исправленной ошибки неизменно отрицательно влияет на репутацию программного продукта и на репутацию коллектива его разработчиков.

Вопрос создания мер по предотвращению подобных ситуаций выходит за рамки предлагаемой статьи. Однако здесь предлагается подход, позволяющий выявить такие повторные появления уже однажды исправленных ошибок на этапе тестирования. Подход этот состоит в проверке каждой версии программного продукта на отсутствие всех ранее исправленных ошибок. Такая проверка позволяет избежать повторного появления ошибочного поведения программы вне зависимости от его причины. Осуществление данного подхода при тестировании коллективом разработчиков требует много времени при выпуске каждой версии. Однако автоматизация этого процесса, как будет показано в статье ниже, делает его вполне осуществимым, и это позволяет значительно повысить надежность программного продукта.

2. Особенности автоматизации тестирования в системах реалистичной графики

Программные ошибки, выявляемые в системах моделирования освещенности и реалистичной компьютерной графики, можно подразделить на четыре основных класса.

Первый класс связан с ошибкой моделирования распространения света в виртуальной сцене. Ошибки проявляются в неверных вычислениях и результатах расчётов, которые можно представить в численном виде. Соответственно, тестирование данного класса ошибок строится на анализе или сравнении численных результатов расчёта. Так как моделирование освещенности часто требует значительного времени вычислений, то для этого класса ошибок важным аспектом является создание эффективных тестовых сцен.

Второй класс связан с ошибкой построения реалистичного изображения или ошибочной визуализацией результатов моделирования. Ошибки проявляются в некорректной картинке или неверном отображении результатов в графическом виде. Тестирование этого класса ошибок требует анализа и сравнения изображений или графических результатов расчёта.

Третий класс ошибок касается эффективности выполнения программы. Он включает в себя как случаи потери скорости расчётов (программа затрачивает на расчёт значительно большее время, чем раньше), так и потери при использовании памяти (программа необоснованно затрачивает при выполнении большее количество оперативной памяти, чем раньше или чем должно быть по оценке). Сюда же можно отнести ошибки «утечки» памяти, когда выполнение алгоритмов с выделением и последующим освобождением памяти приводит к неожидаемому уменьшению свободной памяти. Для автоматической

проверки отсутствия таких ошибок нам необходимо анализировать скорость работы программы и объем используемой памяти.

Наконец четвертый, последний класс ошибок включает в себя «зависания» программы в процессе работы или ее аварийное завершение, вызванные определённым набором входных данных или определёнными действиями пользователя. Проверка отсутствия таких ошибок возможна на основе слежения за состоянием расчётных процессов для определения их «зависания» или аварийного завершения.

Программные продукты, разрабатываемые нашим коллективом, предоставляют необходимую функциональность в нескольких формах. С точки зрения автоматизации тестирования важными являются:

- пакетная обработка данных (или выполнение командной строки);
- API, предоставляемый как пакет для языка программирования Python;
- Интерактивная программа с графическим интерфейсом пользователя;
- Дополнение (plug-in) к системе автоматизированного проектирования CATIA [5].

Все эти программы и приложения созданы на базе общего вычислительного ядра системы с различными модулями интерфейса. Программные ошибки могут проявляться как в одном из приложений, так и во всех формах сразу. Поэтому система тестирования должна поддерживать все указанные функциональности. Если ошибка воспроизводится только в каком-либо одном приложении, то тестирование вынужденно применяется именно для этой формы. Однако для случаев, когда ошибка воспроизводится в нескольких формах, ее тестирование целесообразно выполнять в форме, в которой тест может быть создан и выполнен наиболее эффективно. Например, ошибку в алгоритме трассировки лучей, допущенную внутри общего вычислительного ядра, можно будет выявить независимо от используемого интерфейса. Поэтому тест на отсутствие этой ошибки лучше написать с использованием самой простой формы, позволяющей запустить его на выполнение в фоновом режиме, – пакетной обработки данных.

Каждый автоматический тест выдает решение о том, пройдён ли тест успешно. Если тест не пройден, то возможна выдача дополнительной информации, призванной помочь в нахождении ошибки при анализе результатов тестирования. По окончании работы тестирующей системы выдаётся таблица со всеми результатами, что позволяет сделать вывод либо об успешном прохождении данного этапа автоматического тестирования, либо о необходимости исправления найденных ошибок. Например, для облегчения анализа результатов в случае неудачного теста на ошибку визуализации автоматически создаются два изображения: ожидаемое и полученное, с количественным и качественным описанием разницы между ними.

3. Тесты в фоновом режиме

3.1. Выполнение командной строки

Так как создание теста для командной строки является обычно простой задачей с технической точки зрения, то этот тип тестов является наиболее предпочтительным, если ошибка воспроизводится с его помощью, даже если изначально она была обнаружена при выполнении Python скрипта или в интерфейсной программе. Дополнительным преимуществом такой формы тестов является возможность выполнять их в фоновом режиме. Разработка теста состоит, как правило, в подготовке исходных данных, написании скрипта для выполнения необходимого расчёта и анализа результатов. Численные результаты теста проверяются через сравнение их с результатами, полученными либо аналитически, либо ранее вычисленными предыдущей, заведомо верной версией программы. Графические результаты проверяются через сравнение

полученных изображений с изображениями, полученными предыдущей корректной версией программы.

В наших комплексах в основе моделирования освещенности лежат стохастические методы трассировки лучей или фотонов света, поэтому полученные изображения могут различаться часто незаметно для глаза. Поэтому при сравнении изображений проверяется не полное их совпадение, а допускается отклонение с определённым уровнем точности, величина которого зависит от тестируемой подсистемы. Сравнением изображений занимается специальная программа, выдающая в результате разницу между пикселями сравниваемых изображений. Разница выдаётся в числовом формате (абсолютная и относительная разность между пикселями) и в графическом, в виде изображения, где большая яркость пикселя соответствует большей разности между сравниваемыми изображениями в этой точке.

Пакетный режим также можно использовать для контроля скорости работы программы путём вычисления времени, затраченного модулем на расчёт, и для проверки отсутствия «зависания» и аварийного завершения программы при определённых исходных данных путём контроля наличия и своевременного завершения соответствующих процессов. Однако для этого класса ошибок нельзя использовать выполнение тестов в фоновом режиме.

3.2. Использование Python API

Использование для тестирования скриптов на языке Python с вызовами функций API нашей системы моделирования является методом, следующим по предпочтительности после тестирования в пакетном режиме. Он применяется в тех случаях, когда командной строке не хватает функциональности, или необходимо задавать определенные настройки режима моделирования. Этот подход, за счёт расширенных возможностей программного API [6] и самого языка программирования Python, позволяет автоматизировать действия пользователя по работе с различными объектами комплекса, такими как объекты сцены, модули рендеринга, различные симуляторы и т.д. К настоящему времени общее число подобных объектов весьма велико, например, одних только типов объектов сцены существует более сотни. Также необходимо отметить, что многие из этих объектов имеют достаточно большое количество различных настроек, влияющих на режимы их работы и вычислений.

Использование программного API комплекса позволяет проверить все те же типы ошибок, что и модуль командной строки, а также дополнительно проверить количество памяти, занимаемое программой в процессе работы, отсутствие «зависания» и аварийного завершения программы в результате определённой последовательности команд. Для проверки отсутствия ошибок с «утечкой» памяти проводится определённое число одинаковых операций, захватывающих и потом освобождающих память, и сравнивается объём памяти, занимаемый программой после каждой операции. Если выявлено, что количество занятой памяти после каждой операции постоянно увеличивается, то это однозначно указывает на ошибку.

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, содержащий в себе интерпретатор языка Python с возможностью доступа ко всем остальным объектам комплекса. Пример простейшего скрипта на языке Python приведён на листинге 1.

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

Листинг 1. Пример простейшего скрипта на языке Python
Listing 1. An example of a simple Python script

В данном примере в ядро комплекса 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.

3.3. Многопоточность в сценариях

При автоматическом тестировании компонентов комплекса оптического моделирования весьма важным критерием является эффективность данной процедуры с точки зрения затраченного времени. 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 значительно 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 или 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 в тестовых сценариях является предпочтительным, так как при этом можно добиться большего параллелизма, особенно в задачах пакетной обработки данных. Стоит отметить, что модуль multiprocessing не имеет каких-либо ограничений [8].

Пример многопоточного тестового сценария, накладывающего фильтр на множество отдельных изображений, приведён на листинге 2.. В данном примере каждый 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()
```

Листинг 2. Пример многопоточного тестового сценария
Listing 2. Sample multithreaded test script

4. Тестирование пользовательского интерфейса

4.1. Тестирование интерфейса автономного продукта

Все методы тестирования программного обеспечения можно разделить на две группы: метод "чёрного ящика", когда внутренние алгоритмы работы системы неизвестны тестировщику (или автору автоматического теста), и метод "белого ящика", когда такие алгоритмы известны, и их особенности учитываются при создании автоматического теста. Тесты пользовательского интерфейса в основном относятся к первой группе, так как для пользователя, который работает с интерфейсом, обычно важен только конечный результат работы программы при различных данных, которые он вводит с его помощью. Тестирование с использование графического интерфейса обычно наиболее трудозатратно как при создании теста, так при его выполнении. Поэтому его используют, когда другие тесты не позволяют выявить ошибку.

Много лет назад нашим коллективом была разработана система записи и воспроизведения интерактивных действий пользователя [9], которая позволяла полностью автоматизировать процесс тестирования пользовательского интерфейса для нашего комплекса оптического моделирования. Однако с переводом всех интерфейсных программ на платформу Qt использование этой системы без серьезной переработки стало невозможным. Поэтому мы сочли, что будет более эффективно взять один из существующих пакетов записи действий в интерактивной программе. Нами были сформулированы требования к такому пакету:

- способность распознавать элементы пользовательского интерфейса (окна, кнопки, поля ввода, диалоги, слайдеры и т.д.) и производить над ними действия (ввод символов, нажатия кнопками мыши и т.д.);
- поддержка языка сценариев, позволяющего производить математические операции над числами и операции над текстовыми строками;
- возможность модификации сценария без его полной перезаписи;
- чтение и запись текстовых файлов;
- работа с относительными координатами в окне тестируемой программы вместо работы с экранными координатами;
- отсутствие необходимости изменения кодов тестируемой программы (например, включение вспомогательных кодов для взаимодействия с внешним тестирующим пакетом);
- способность получать информацию о выполняемых процессах;
- наличие операций с таймерами.

Авторами были протестированы более семидесяти программных пакетов, позволяющих в том или ином виде записать и воспроизвести действия пользователя в интерактивной программе. В результате был выбран пакет AutoIt [10], удовлетворяющий большинству

условий. Описание некоторых протестированных пакетов, включая AutoIt, приведены в [11]. Для тестирования с использованием пакета AutoIt создаются сценарии на проверку каждой функциональности тестируемой программы или известной ошибки. Сценарий выполняется, воздействуя на элементы пользовательского интерфейса этой программы, и результат выполнения сценария сравнивается с ожидаемым. Тест считается успешным, если результат моделирования или генерации изображения совпадают с корректным результатом (либо полученным теоретически, либо вычисленным более ранней и заведомо корректной версией программы), или если наступает ожидаемая реакция программы (например, нажатие кнопки интерфейса, сообщение об ошибке и т.д.) и отсутствует ошибочная реакция программы (например, аварийное завершение).

Неудачное завершение теста не всегда говорит об ошибке в программе, оно также бывает и при изменении функциональности. Нередко исправление одной ошибки отражается сразу на нескольких тестах. В каждом таком случае необходимо провести тщательный анализ результатов и по его итогам либо исправить выявленную ошибку, либо доработать тест (например, изменить корректное эталонное изображение или порядок действий пользователя).

После значительных изменений расположения элементов пользовательского интерфейса в окнах программы тест также может завершиться неудачно. В этом случае выдается диагностика «невозможно найти требуемый элемент интерфейса». Если элемент пользовательского интерфейса был удален, то тест становится бесполезным, и мы должны отказаться от его использования. Если же элемент был перемещен, то можно изменить только кусок теста с учётом изменённого расположения элемента. Такое локальное изменение занимает обычно несколько минут. Надо отметить, что в большинстве случаев такие изменения не требуются, т.к. интерактивные тесты создаются максимально гибко и обычно являются устойчивыми к перемещению и добавлению элементов интерфейса в пределах одного диалогового окна.

К сожалению, использовать пакет AutoIt не просто. В нём не полностью реализована поддержка пакета GUI SDK «Qt», что приводит к неудобствам в работе. Например, отсутствует поддержка элементов меню, нет возможности захвата изображения с области экрана для дальнейшей обработки или записи в файл, нет возможности узнать состояние некоторых элементов интерфейса (например, текущий элемент, выбранный в выпадающем списке). Симуляция интерфейсных операций, даже таких простых как, например, нажатий клавиш и движения мыши, потребовало дополнительных усилий, чтобы сделать их максимально точными и надёжными во всех поддерживаемых версиях Windows. Приведем примеры некоторых решений, разработанных под AutoIt.

4.2. Решения, разработанные под AutoIt

Модульный подход, применённый в процессе разработки программных комплексов, позволяет использовать одни и те же GUI элементы в различных местах пользовательского интерфейса. По этой причине оказалось целесообразным создание библиотеки функций на языке сценариев AutoIt для ускорения создания тестов. В таком случае модулям (диалогам) пользовательского интерфейса естественным образом соответствуют модули (функции) на языке сценариев AutoIt, такие как «открыть файл», сохранить результат расчёта» и другие. Например, на листинге 3 показано, как выглядит на этом языке сценариев функция для сохранения сцены – модели данных, состоящей из многих объектов различных типов, связанных определёнными иерархическими связями (геометрические объекты, свойства их поверхностей, источники света, и т.д.), открытой в приложении в данный момент:

```
; Function to save current scene under given filename
Func SaveAs($file)
```



```
  ; Open menu File -> Save As
  Send("!fs{ENTER}")

  ; Wait for dialog to appear
  If WinWaitActive("Save Scene As...", "", $Timeout) = 0 Then
    Report("'Save scene As' window not found")
    Exit(1)
  EndIf

  ; Type filename in the dialog and press ENTER
  Send($file & "{ENTER}")

  ; Close dialog
  If WinWaitClose("Save Scene As...", "", $Timeout) = 0 Then
    Report("'Save scene As' window could not be closed")
    Exit(1)
  EndIf
EndFunc
```

Листинг 3. Функция для сохранения сцены на языке сценариев
Listing 3. Function to save a scene in a scripting language

Часто используемые сложные операции, такие как, например, присваивание геометрическому объекту сцены заданное оптическое свойство, также включены в библиотеку функций на языке сценариев AutoIt. В качестве примера можно привести загрузку из файла данных о дисторсии камеры. Эта функция в свою очередь использует другую функцию это библиотеки, `OpenCamProp()`, для открытия диалога свойств камеры.

```
; Function to load radial distortion data into camera
Func LoadCamDistr($file)
  ControlClick("[ACTIVE]", "", "[TEXT:BrowserWindow]")
  Sleep(500)
  ; Open "Camera properties" dialog
  OpenCamProp()

  ControlClick("Camera Properties", "", "[TEXT:rd_browse_pbtn]")
  ; Type filename
  Send($file & "{ENTER}")
  ControlClick("Camera Properties", "", "[TEXT:ok_btn]")
  Sleep(500)
EndFunc
```

Листинг 4. Функция загрузки из файла данных о дисторсии камеры
Listing 4. Function to load from file data about camera distortion

4.3. Тестирование пользовательского интерфейса в САПР CATIA

Тестирование дополнительного модуля (plugin) в системе автоматизации проектирования CATIA имеет свои отличия, связанные с особенностями пользовательского интерфейса системы. Благодаря наличию общего вычислительного ядра всех наших продуктов тестирование процесса моделирования выполняется автономно, методами описанными ранее.

Пользовательский интерфейс CATIA отличается от интерфейса других подобных систем тем, что дерево объектов сцены отображается в том же окне, что и сама сцена [12]. Стандартное окно системы CATIA с отображённым в нем деревом сцены показано на рис. 1. Ни само дерево, ни его отдельные элементы не имеют внешних идентификаторов. Поэтому в любом случае необходимы некоторые дополнительные действия для

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

Первый способ предусматривает выделение мышью определённой точки на элементе интерфейса, это ведет к фиксации конкретных координат этой точки в сценарии теста. Второй способ заключается в использовании алгоритмов распознавания изображений. Оба этих способа имеют свои недостатки: в первом случае – необходимость использования строго определённого разрешения экрана и масштаба элементов управления, во втором случае – необходимость учитывать возможную ошибку распознавания, особенно, если на экране присутствуют несколько похожих друг на друга объектов.

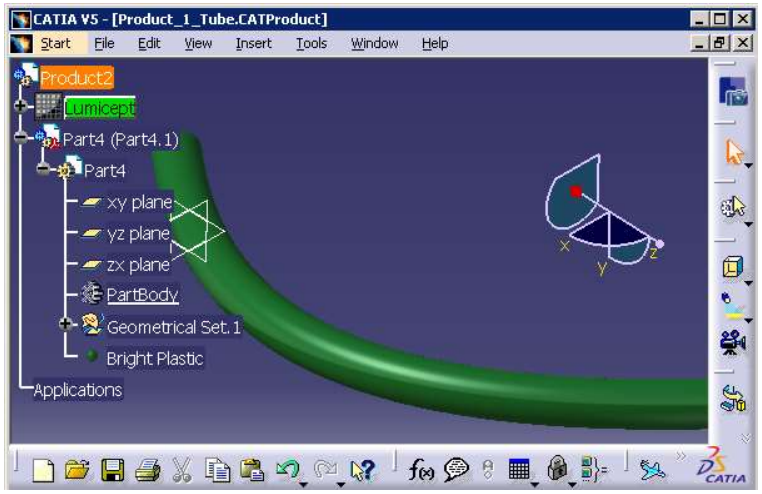


Рис. 1. Окно системы CATIA с загруженной сценой и отображённым деревом иерархии сцены
Fig. 1. CATIA system window with a loaded scene and the displayed scene hierarchy tree

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 (AutoIt в нашем случае) 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, но при создании нашего plugin-а оптического моделирования к нему были добавлены дополнительные параметры. При тестировании используется сценарий на языке CATVBs (разновидность языка Visual Basic), который вызывается из самой системы, а также внешний инструмент AutoIt и его встроенный язык сценариев.

Тестирование состоит из следующих этапов.

1. Система CATIA запускается в стандартном режиме с графической оболочкой и с автоматическим запуском сценария. Для запуска используется пакетный командный файл, в котором заданы параметры – необходимая оболочка (Workbench) системы CATIA, а также путь к сценарию, который будет выполнен 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. Выполняется первый сценарий на языке CATVBs, работающий с 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 ждёт завершения работы программы AutoIt, на протяжении этого времени диалог параметров остаётся открытым. Его закрытие произойдёт уже далее, с помощью сценария AutoIt, который будет имитировать нажатие пользователем кнопки «OK». После закрытия диалога параметров управление снова переходит к первому сценарию, который считывает значения требуемых параметров из объекта материала и проверяет, что они были правильно заданы с помощью диалога параметров.

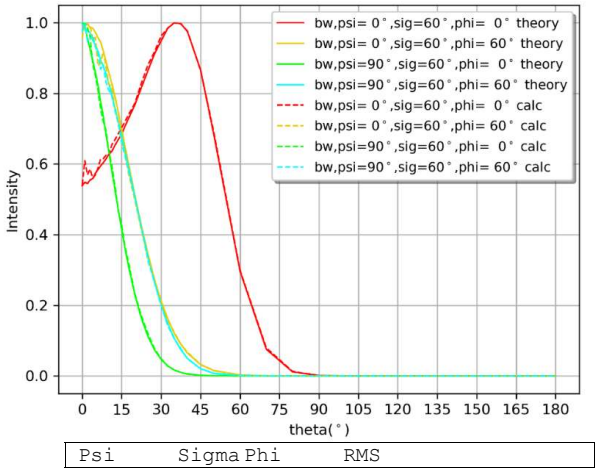
3. Выполнение сценария программы AutoIt. Этот сценарий находит диалоговое окно с заголовком "Properties" и проверяет, что оно принадлежит системе CATIA с помощью сравнения идентификаторов главного окна системы CATIA и данного диалогового окна. Затем AutoIt производит активацию этого окна, выполняет смысловые действия теста (например, записывает в определенное поле текст) и завершает работу с окном (например, инициирует нажатие кнопки «OK»). Управление снова передаётся самой системе CATIA, в которой продолжает работу первый сценарий. Он проверяет правильность установленных данных.

Этот пример обеспечивает оптимальное сочетание принудительной активации тех элементов пользовательского интерфейса, тестирование которых требуется обязательно обеспечить, и воздействия на объекты сцены с помощью встроенных сценариев в тех случаях, когда активация элементов пользовательского интерфейса затруднена.

5. *Примеры автоматизированных тестов*

Ниже приведены примеры результатов тестирования с использованием различных критериев их правильности: корректность вычисленных величин, корректность визуализации, определяемая через разность эталонного и вычисленного изображений.

Тест 1. Результат теста на совпадение результатов расчёта с заранее рассчитанными аналитическими значениями выглядит так, как показано на *рис. 2*.



```
 0.0    60.0     0.0    0.22% passed
 0.0    60.0    60.0    0.34% passed
90.0    60.0     0.0    0.16% passed
90.0    60.0    60.0    0.17% passed
---------------------------------------
Status: passed
```

Рис. 2. Сравнение полученных результатов с аналитическими
Fig. 2. Comparison of the obtained results with analytical ones

Строятся графики аналитических и вычисленных значений, вычисляются среднеквадратичные отклонения (RMS), по которым и принимается решение о прохождении теста.

Тест 2. На рис. 3 приведены эталонное изображение наложения текстуры, полученное методом BRT (Backward Ray Tracing – обратная трассировка лучей) в предыдущей версии программы, и изображение, полученное тем же методом в текущей (проверяемой) версии.

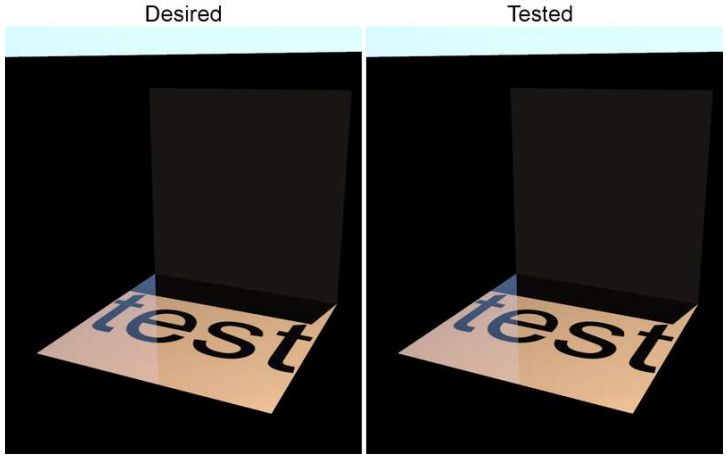


Рис. 3. BRT, наложение текстуры
Fig. 3. BRT, texture mapping

Для положительного прохождения теста разница между изображениями не должна превышать определённого значения:

```
RMS between desired and actual images = 0.0005%
-----------------------------------------------
Status: passed
```

Тест 3. В этом тесте проверяется правильность моделирования естественного освещения в методе Path Tracing (трассировка путей). На рис. 4 приведены эталонное изображение объекта при дневном свете, полученное в предыдущей версии программы, и изображение, полученное тем же методом в текущей (проверяемой) версии.

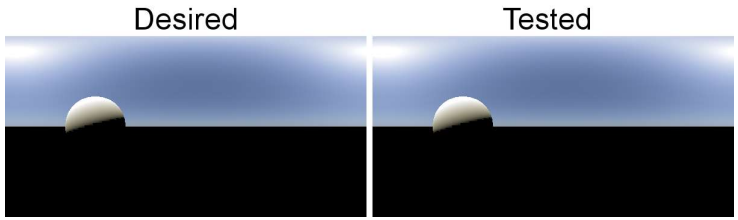


Рис. 4. Path Tracing, изображение при дневном освещении
Fig. 4. Path Tracing, daylight image

Так же, как и в предыдущем случае, разница изображений в виде среднеквадратичного отклонения используется для автоматического принятия решения о прохождении теста. Сами изображения сохраняются для возможного последующего анализа в случае отрицательного решения.

Тест 4. Следующий пример – сравнение двух изображений, полученных с использованием различных технологий трассировки лучей. На рис. 5 показано изображение, полученное методом BRT, на рис. 6 изображение получено методом PT.

Рис. 5. Метод Backward Ray Tracing
Fig. 5. Backward Ray Tracing Method

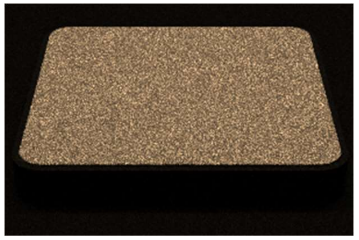


Рис. 6. Метод Path Tracing
Fig 6. Path Tracing Method

Визуально изображения различаются, это обусловлено особенностями использованных методов. Главное отличие в том, что BRT – детерминированная трассировка лучей, а PT – стохастическая, ее изображение содержит стохастический шум, неизбежный при малом времени вычислений. Однако среднее значения цвета и яркости должны быть одинаковы, вернее, не превышать определённого значения. Таким тестом производится перекрёстная проверка правильности разных методов трассировки.

```
Relative error between color of BRT and PT images = 0.2666%
-----------------------------------------------------------
Status: passed
```

Тест 5. Этот пример иллюстрирует подход, позволяющий одновременного тестировать нескольких функций программного продукта за один шаг (рис. 7). В специально созданной тестовой сцене проверяется правильность использования сложных атрибутов поверхностей, задание дневного освещения в различных методах моделирования (уже упомянутые BRT и PT, а также MCRT и BMCRT – прямая и обратная стохастическая трассировка лучей). Также здесь тестируется отображение значений яркостей в псевдоцветах. Для анализа прохождения теста, как и прежде, используется сравнение вновь полученного изображения с эталонным, полученным предыдущей версией программы, а решение принимается по среднеквадратичному отклонению.

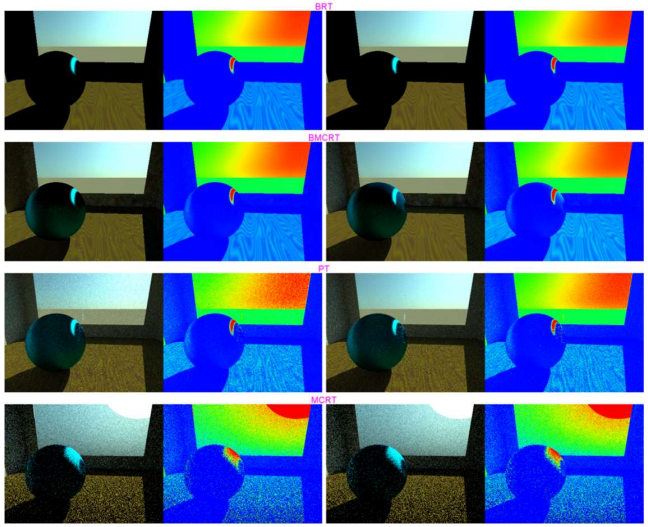


Рис. 7. Проверка корректности использования атрибутов поверхностей
Fig. 7. Verification of the use of surface attributes

Рис. 8. Эталонное изображение
Fig. 8. Reference image

Рис. 9. Изображение, полученное от тестируемой программы
Fig. 9. Image received from the tested program

Рис. 10. Разность изображений
Fig. 10. Image difference

Тест 6. Этот тест также сравнивает изображение, выдаваемое тестируемой версией программы, с эталонным изображением, полученным ранее. Тестируются самосветящиеся объекты сцены, используемые в качестве источников света. Визуально разница в изображениях незаметна: трудно сказать, насколько отличаются изображения на рис. 8 и рис. 9. Для анализа вычисляется разница между двумя изображениями и выдается в виде абсолютных значений и в виде изображения, удобном для восприятия (рис. 10).

На рис. 10 различными цветами отображается разность абсолютных значений яркости в соответствующих пикселях исходных изображений. Максимальное абсолютное значение разницы между эталонным и реальным изображениями для данной сцены составило 1.81%.

6. Заключение

Описанный подход позволяет выявить повторное появление в программных продуктах однажды уже исправленных ошибок. Важным фактором является правило, когда процесс исправления каждой обнаруженной и зарегистрированной ошибки обязательно завершается созданием очередного автоматического теста. Очевидно, что, помимо непосредственного выявления повторных ошибок, такое тестирование помогает заблаговременно выявить также и новые ошибки, которые оказывают влияние на результат моделирования или поведение программы, проверяемые имеющимися тестами. Это обусловлено тем, что в формировании таких результатов, как правило, используется работа сразу нескольких программных компонент, и наличие ошибки в любой из них может повлиять на конечный результат расчёта.

Автоматизация такого регрессионного тестирования открывает саму возможность интенсивного тестирования каждой версии выпускаемого продукта, что, безусловно, положительно влияет на его надежность. Выполнение каждого теста, даже если он тщательно разработан и записан, требует от человека прочитать и вспомнить, что и как необходимо проверить в данном тесте, а после выполнения – проанализировать результаты. В то же время автоматическая система выполняет тесты один за другим, без остановок, что позволяет значительно сократить время, затрачиваемое на тестирование программных продуктов. В случае необходимости время тестирования продукта можно уменьшить путём использования параллельно нескольких компьютеров, каждый из которых будет выполнять свой набор тестов независимо друг от друга. Таким образом, процесс интенсивного тестирования может быть включен в план выпуска версии продукта даже при условии жестких сроков его выпуска.

Описанная автоматическая система регрессионного тестирования используется при разработке и поддержке программного комплекса оптического моделирования и реалистичной компьютерной графики Lumicept [13, 14], существующего как в виде автономного продукта, так и дополнения (plugin) к широко используемой в автомобильной и авиационной промышленности САПР CATIA. Она приводит к существенной экономии времени и усилий разработчиков, позволяя повысить надежность продукта, минимизируя затраты. За несколько лет был создан набор из ~250 автоматизированных тестов. Автоматическое выполнение всего этого набора занимает около четырех часов на компьютере Xeon E3-1275 v5 3.60 GHz, в то время как оценка времени, необходимого для подобного тестирования человеком, составляет две-три недели.

Список литературы / References

[1]. Bouquet F., Grandpierre C., Legeard B., Peureux F. A test generation solution to automate software testing. In Proc. of the 3rd International Workshop on Automation of Software Test, 2008, pp.45-48.

[2]. Гленфорд Майерс, Том Баджетт, Кори Сандлер. Искусство тестирования программ, 3-е издание, М., «Диалектика», 2012 г., 272 стр. / Glenford J. Myers, Tom Badgett, Corey Sandler. The Art of Software Testing, 3rd Edition. Wiley, 2011, 256 p.

[3]. Tretmans J., Belinfante A. Automatic Testing with Formal Methods. In Proc. 7th of the European International Conference on Software Testing, Analysis and Review, 1999, pp.2012-2012

[4]. Huang Z., Carter L. Automated solutions: Improving the efficiency of software testing. Issues in Information Systems, vol. 4, 2003, pp. 171-177.

[5]. Барладян Б.Х., Волобой А.Г., Галактионов В.А., Шапиро Л.З. Интеграция реалистичной графики в системы автоматизированного проектирования и управления жизненным циклом изделия. Программирование, том 44, no. 4, 2018 г., стр. 26-35 / Barladian B.Kh., Voloboy A.G., Galaktionov V.A., Shapiro L.Z. Integration of Realistic Computer Graphics into Computer-Aided Design and Product Lifecycle Management Systems. Programming and Computer Software, vol. 44, no. 4, 2018, pp. 225–232. DOI: 10.1134/S0361768818040047

[6]. Дерябин Н.Б., Жданов Д.Д., Соколов В.Г. Внедрение языка сценариев в программные комплексы оптического моделирования. Программирование, vol. 43, no. 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, no. 1, 2017, pp. 13-23. DOI: 10.1134/S0361768817010029.

[7]. Van der Walt S., Schönberger J. L., Nunez-Iglesias J., Boulogne F., Warner J. D., Yager N., Yu T. Scikit-image: image processing in Python. PeerJ, vol. 2, 2014, article e453.

[8]. Singh N., Browne L. M., Butler R. Parallel astronomical data processing with Python: Recipes for multicore machines. Astronomy and Computing, vol. 2, 2013, pp. 1-10.

[9]. Волобой А.Г., Денисов Е.Ю., Барладян Б.Х. Тестирование систем моделирования освещенности и синтеза реалистичных изображений. Программирование, vol. 40, no. 4, 2014 г., стр. 13-22 / Voloboi A. G., Denisov E. Yu., Barladyan B. Kh. Testing of Systems for Illumination Simulation and Synthesis of Realistic Images. Programming and Computer Software, vol. 40, no. 4, 2014, pp. 166–173. DOI:10.1134/S0361768814040094.

[10]. AutoIt. Available at: http://www.autoitscript.com, accessed: 20.02.2020.

[11]. Денисов Е.Ю., Волобой А.Г., Калугина И.А. Автоматизация тестирования интерактивной системы моделирования освещенности. Препринты ИПМ им. М.В. Келдыша, № 200, 2018, 19 стр. / Denisov E. Yu., Voloboi A. G., Kalugina I.A. Automatic testing of interactive lighting simulation software package. Keldysh Institute preprints, 2018, 200, 19 p. (in Russian).

[12]. Dassault Systemes, Inc. CATIA Version 5-6 R2015 Documentation. Available at: http://media.3ds.com, accessed 05.08.2018.

[13]. Жданов Д.Д., Потемин И.С., Галактионов В.А., Барладян Б.Х., Востряков К.А., Шапиро Л.З. Спектральная трассировка лучей в задачах построения фотореалистичных изображений. Программирование, vol. 37, no. 5, 2011 г., стр. 13-26. / Zhdanov D. D., Potemin I. S., Galaktionov V. A., Barladyan B. Kh., Vostryakov K. A., Shapiro L.Z. Spectral Ray Tracing in Problems of Photorealistic Imagery Construction. Programming and Computer Software, vol. 37, no. 5, 2011, pp. 236–244. DOI: 10.1134/S0361768811050069.

[14]. Жданов Д.Д., Гарбуль А.А., Потемин И.С., Волобой А.Г., Галактионов В.А., Ершов С.В., Соколов В.Г. Фотореалистичная модель объемного рассеивания в задаче двунаправленной стохастической трассировки лучей. Программирование, vol. 41. No. 5, 2015 г., стр. 66-75. / Zhdanov D.D., Garbul A.A., Potemin I.S., Voloboy A.G., Galaktionov V.A., Ershov S.V., Sokolov V.G. Photorealistic Volume Scattering Model in the Bidirectional Stochastic Ray Tracing Problem. Programming and Computer Software, vol. 41, no. 5, 2015, pp. 295–301, DOI: 10.1134/S0361768815050102.

Информация об авторах / Information about authors

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.

Алексей Геннадьевич ВОЛОБОЙ, ведущий научный сотрудник, доктор физико-математических наук, доцент. Научные интересы: компьютерная графика, оптика, оптическое моделирование.

Alexey Gennadievich VOLOBOY, Leading Researcher, Doctor of Physical and Mathematical Sciences, Associate Professor. Research interests: computer graphics, optics, optical modeling.

Елисей Дмитриевич БИРЮКОВ – младший научный сотрудник. Его научные интересы включают компьютерную графику, вычислительную оптику, трассировку лучей.

Elisey Dmitrievich BIRUKOV, junior researcher. His research interests include computer graphics, computing optics, ray tracing techniques.

Михаил Сергеевич КОПЫЛОВ – старший инженер. Его научные интересы включают компьютерную графику, вычислительную оптику, трассировку лучей.

Mikhail Sergeevich KOPYLOV, senior engineer. His research interests include computer graphics, computing optics, ray tracing techniques.

Ирина Александровна КАЛУГИНА является младшим научным сотрудником. Её научные интересы включают компьютерную графику, моделирование освещённости.

Irina Alexandrovna KALUGINA is a junior researcher. Her research interests include computer graphics, lighting simulation.