Санкт-Петербургский Государственный Университет

Программная инженерия
Кафедра системного программирования

Орачев Егор Станиславович

Разработка библиотеки для визуализации 3D графики

Курсовая работа

Научный руководитель:
ст. преп. А. А. Пименов

Санкт-Петербург
2020

Оглавление

Введение

Трехмерная графика — раздел компьютерной графики, посвященный методам визуализации объемных моделей и объектов, описанных в 3D (3-dimensional) пространстве. К основным областям применения трехмерной графики можно отнести: системы автоматизированного проектирования, кинематограф, анимацию, видеоигры, симуляторы и многое другое.

Существует несколько способов построения изображения по трехмерным данным, одними из которых являются: растеризация примитивов, трассировка лучей, трассировка пути. Применение конкретного способа с последующей его конфигурацией обуславливается различными факторами:

- Временными ресурсами. При работе в режиме реального времени максимальный интервал генерации изображения составляет приблизительно 33 миллисекунды (соответствует частоте обновления 30 Гц), что позволяет создать непрерывный для человеческого глаза видеопоток.

- Вычислительными ресурсами, предоставляемыми конкретной платформой, вместе со спецификой их использования. К примеру, мобильные устройства накладывают существенные ограничения на энергопотребление и не имеют активное охлаждение.

- Объемом отображаемых данных.

- Качеством отображения, которое может характеризоваться как фото-реалистичное, физически корректное, кинематографичное, "мультяшное" [14] и так далее.

Наличие вышеизложенных факторов в сочетании с многочисленными требованиями реальных проектов, связанных с визуализацией данных, создает ряд проблем при поиске уже готовых графических решений и при их непосредственной интеграции в систему.

В проекте CoreCVS [3] (Computer Vision Primitives Library) в рамках работы над библиотекой для симуляции физики полетов дронов разработчики столкнулись с необходимость визуализации процесса симуляции как для отладки программ, так и для лучшего понимания того, как взаимодействуют объекты между собой. Для решения этой проблемы используется существующая в рамках проекта графическая библиотека, написанная с использованием OpenGL [11] (Open Graphics Library), которая позволяет отображать относительно простую геометрию с базовой моделью освещения. Однако данная библиотека использует в своей основе уже устаревшее подмножество функциональности OpenGL, что создает необходимость значительного рефакторинга для дальнейшего ее использования.

В итоге было принято решение разработать с нуля новую графическую библиотеку в рамках CoreCVS, которая не только бы замещала функциональность своей предшественницы, но и расширяла ее в соответствии с задачами, решаемыми как в текущих, так и в будущих проектах.

Данное решение мотивированно не только стремлением минимизировать количество сторонних зависимостей всего проекта, но и желанием использовать в качестве основы новейшее Vulkan API [13] (Vulkan Application Programming Interface), которое призвано заменить OpenGL в силу его морального устаревания. Анализ существующих решений также показал невозможность интеграции сторонних библиотек ввиду ряда причин, изложенных далее в тексте работы.

1. Постановка задачи

Цель данной работы — реализация библиотеки для визуализации трехмерной графики в реальном масштабе времени с возможностью использовать как низкоуровневый интерфейс для создания специфичных моделей отображения, так и высокоуровневые абстракции, предоставляющие стандартный способ визуализации объектов. Для ее достижения были выделены следующие задачи:

- Исследование предметной области. Сравнение существующих решений
- Разработка и формализация требований к проекту
- Реализация компонентов графической библиотеки с учетом заявленных требований
- Реализация поддержки Vulkan API

2. Терминология

- Графическая подсистема (англ. Graphical backend) — модуль программы, реализующий графическую функциональность приложения.
- Шейдер (англ. Shader) — программа, предназначенная для исполнения на графическом процессоре.
- Материал — контейнер, которых хранит описание поверхности объекта в виде набора атрибутов и некоторую программу, которая контролирует процесс обработки данного материала.
- Графический интерфейс (англ. Graphical API) — интерфейс для взаимодействия с графическим ускорителем системы.
- Application Programming Interface (API) — интерфейс прикладного программирования.
- Central Processing Unit (CPU) — центральный процессор.
- Graphical Processing Unit (GPU) — графический процессор.
- Graphical User Interface (GUI) — графический пользовательский интерфейс.

3. Требования к проекту

Основные функциональные требования, которые вытекают прежде всего из задач, решаемых в рамках проекта CoreCVS, изложены далее:

- Генерация изображения в режиме реального времени: в своей основе библиотека полагается на алгоритм растеризации геометрии с использованием *Z-буфера* [1], что позволяет при должном количестве геометрии и вычислительных ресурсах генерировать изображения с частотой, достаточной для создания псевдонепрерывного видеопотока.

- Кроссплатформенность: библиотека компилируется и работает на линейке основных операционных систем (Window, Linux, MacOS).

- Независимость от графического интерфейса: библиотека взаимодействует с видеоускорителем системы через унифицированный интерфейс, который сам по себе является частью проекта.

- Поддержка Vulkan: доступ к данному API осуществляется через интерфейс, представленный в предыдущем пункте.

- Независимость от структуры сцены: библиотека поддерживает список видимых объектов и осуществляет их последовательное отображение. Иерархию сцены, зависимости между объектами определят пользователь.

- Система материалов: материалы предоставляют типобезопасный интерфейс для конфигурации параметров отображения на GPU.

- Система хранения геометрии: возможность создавать и использовать геометрические объекты с произвольным набором атрибутов.

- Независимость от оконной системы: финальное изображение выводиться на экран, в виджет GUI системы, либо сохраняется во внеэкранном буфере.

4. Существующие решения

В данном разделе проводится обзор и сравнение существующих графических решений с открытым исходным кодом, доступным для свободного использования. Критерии обзора вытекают прежде всего из требований, изложенных ранее. Основные факторы для включения в рассмотрение: наличие документации и обучающих материалов, активная поддержка или разработка, интеграция с Vulkan, поддержка широкого класса графических интерфейсов, индексация[1] на хостинге GitHub.

4.1. BSF

BSF (bs framework) — кросс-платформенная C++14 библиотека, разрабатываемая Game Foundry [2]. Предоставляет унифицированную базу для разработки графических приложений реального времени, видеоигр и инструментов программирования. Поддерживаемые графические интерфейсы: Vulkan, OpenGL, DirectX. Функциональность: конвейер экранных пост-эффектов, отложенное, физически корректное затенение, каскадная генерация теней, система материалов.

4.2. Ogre3D

Ogre3D (Object-Oriented Graphics Rendering Engine) — кросс-платформенна
C++ графическая библиотека, разрабатываемая открытым сообществом [10].
В качестве графических интерфейсов поддерживает OpenGL, DirectX,
WebGL. Функциональность: система материалов и пост-эффектов, менеджмент ресурсов, система анимации, граф сцены.

4.3. Filament

Filament — кросс-платформенная C++ библиотека графики, неофициально поддерживаемая Google [5] и открытым сообществом разработчиков. Спроектирована таким образом, чтобы быть максимально

[1] Дата обращения: 11.12.2019

компактной и эффективной для Android платформы. Поддерживает OpenGL, Vulkan API, WebGL. Предоставляет C++, Java, JavaScript интерфейсы. Основные графические возможности: физически корректное затенение, освещение на основе изображения, система материалов, камера с имитацией физических параметров параметрами, базовые постэффекты.

4.4. Концептуальные недостатки

Несмотря на многочисленную функциональность, которой обладают рассмотренные решения, они не в полной мере удовлетворяют требованиям проекта CoreCVS. В частности, BFS и Ogre3D прежде всего ориентированны на разработку видео игр, поэтому данные библиотеки сильно зависят как от иерархической структуры сцены, так и от обработки данных. Они предоставляют собственные форматы геометрии, языки описания шейдеров и методы загрузки ресурсов, таких как модели, текстуры. Данная функциональность избыточна, так как в проекте CoreCVS уже существуют модули по обработке 3D геометрии и изображений.

Рис. 1: Сравнение существующих решений по основным критериям

Критерии сравнения	**Название библиотеки / проекта**		
	BSF	*Ogre 3D*	*Filament*
Поддержка Vulkan API	да	в процессе реализации	да
Активная поддержка	да	да	да
Наличие документации	да	да	да
Формат геометрии	с фиксированным числом атрибутов	с фиксированным числом атрибутов	с фиксированным числом атрибутов
Структура сцены	иерархическая (компонентная)	иерархическая (компонентная)	иерархическая (настраиваемая)
Форматы ресурсов	специфичные	специфичные	специфичные

Проект Filament имеет больший фокус на графическую составляющую, однако он в первую очередь ориентирован на создание физически корректного освещения на Android платформе, что выражается с существенно меньшей гибкости в настройке как самого освещения, так и экранных эффектов, создание которых практически ограничено на мобильных устройствах ввиду их высокой вычислительной сложности.

Детальное сравнение рассмотренных решений по основным критериям представлено на рис. 1. Красным цветом выделены критические аспекты, желтым — устранимые, но нежелательные особенности, зеленым — полностью приемлемые функциональные возможности.

Поэтому было принято решение отказаться от интеграции одного из существующих решений и сосредоточиться на разработке новой библиотеки, с учетом всей функциональности, которая в той или иной форме уже представлена в проекте CoreCVS.

5. Vulkan API

Vulkan — это кросс-платформенное API для работы с трехмерной графикой и параллельными вычислениями. Предоставляет C99 совместимый интерфейс для взаимодействия с GPU и спецификацию, которая описывает семантику исполнения программ. Впервые был представлен в феврале 2016 года. В настоящее время поддерживается консорциумом Khronos Group.

5.1. Обзор ключевых особенностей

Vulkan является одним из представителей таких графических интерфейсов нового поколения, как Metal [9] и Direct3D [4]. В сравнении со стандартами прошлого поколения, Vulkan имеет ряд новых функциональных особенностей, которые призваны повысить эффективность работы конечных приложений:

- Возможность работать на множестве операционных систем, включаю мобильную платформу.
- Меньшее количество ресурсов, используемых во время работы. Это достигается ручным управлением памятью, используемой драйвером как на CPU, так и на GPU, явным созданием и удалением графических конвейеров обработки примитивов.
- Лучшая расширяемость на платформах с несколькими ядрами. Драйвер позволяет формировать команды для GPU из нескольких потоков.
- Универсальное представление GPU программ в бинарном формате SPIR-V [12]. Это позволяет создавать шейдерные программы на любом из доступных языков, включая GLSL [6] и HLSL [7], и компилировать их в объектные модули независимо от конечной платформы.

- Унифицированный менеджмент вычислительных ядер на GPU. Это позволяет использовать вычислительный конвейер посредством Vulkan API, не прибегая к сторонним вычислительным библиотекам, таким как Nvidia CUDA и OpenCL.

5.2. Сравнение с другими графическими API

В области разработки графических приложений существует ряд промышленных API, часть из которых уже была упомянута ранее в тексте работы. Direct3D предоставляет схожую графическую функциональность, что и Vulkan, однако он доступен только на платформе от компании Microsoft. Графический API Metal, разрабатываемый корпорацией Apple, также доступен только на устройствах и системах этой компании. OpenGL является открытым стандартом, доступным на многих платформах, исключая мобильные устройства, где используется OpenGLES. Однако развитие данного API затруднено требования обратной совместимости, а поддержка его новых версий уже не осуществляется на некоторых устройствах[2].

Таким образом, было принято решение использовать Vulkan API в качестве основы для реализации данного проекта, так как он удовлетворяет функциональным требованиям разрабатываемой библиотеки и обладает активной поддержкой на основных платформах.

[2]Компания Apple прекратила поддержку новых версий OpenGL в 2018, объявив официально данный API устаревшим на своей платформе

6. Описание реализации

В данном разделе приводится описание общей структуры проекта и отдельных компонентов графической библиотеки, реализация которых осуществлялась в рамках данной работы. Исходный код проекта опубликован на хостинге GitHub[3]. Работа по реализации библиотеки выполнялась совместно со студентом 371 группы Сультимом Цырендашиевым.

6.1. Структура проекта

Основные компоненты и классы разработанной графической библиотеки библиотеки представлены на рис. 2.

В качестве языка программирования для разработки использовался C++11, для управления процессом сборки и подключения сторонних зависимостей использовался CMake 3.14, что соответствует требованиям проекта и условиям интеграции с библиотекой CoreCVS.

Для поддержки сборки проекта для трех основных платформ была настроена непрерывная интеграция на платформе Github Actions.

6.2. Поддержка Vulkan API

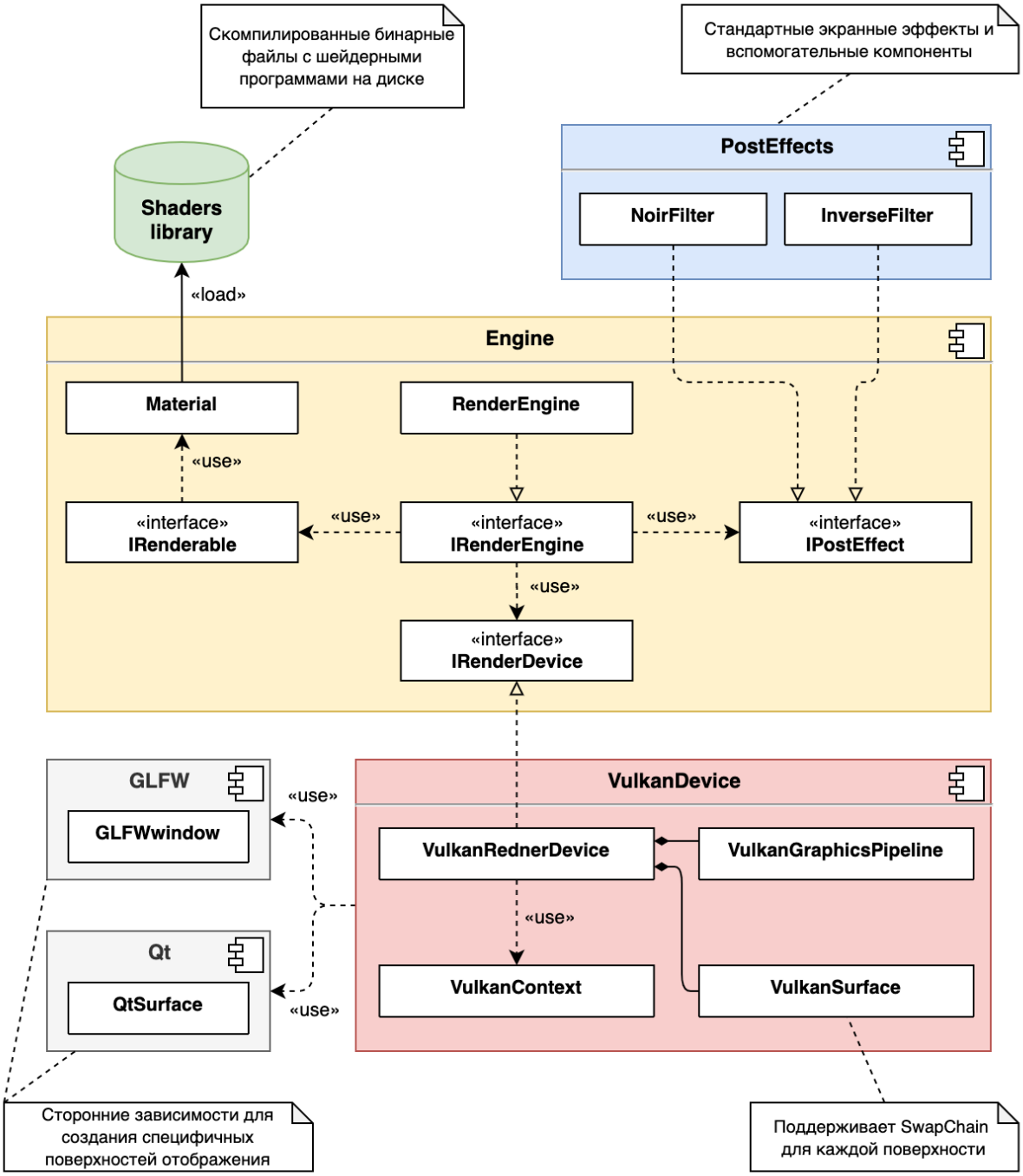
Для поддержки доступа к GPU с использованием API Vulkan был разработан модуль VulkanDevice, который реализует интерфейс абстрактного графического девайса и полностью инкапсулирует логику взаимодействия с этим API.

В рамках данной работы были реализованы следующие компоненты модуля VulkanDevice:

- **VulkanContex**: контекст графической библиотеки Vulkan, который инкапсулирует логику инициализации библиотеки, выбора и конфигурации графического девайса, а также расширений, тре-

[3]Исходный код проекта: https://github.com/EgorOrachyov/Ignimbrite

Рис. 2: UML диаграмма основных компонентов и классов разработанной графической библиотеки



буемых для создания приложения с использованием Vulkan для конкретной операционной системы.

- **VulkanSwapchain**: объект, который инкапсулирует логику презентации финального изображения, сгенерированного на GPU, в специфичное окно или поверхность операционной системы. API Vulkan требует осуществлять явную синхронизации CPU и GPU для доступа к изображению, а также выполнять обработку событий изменения поверхности отображения или конфигурацию *режима презентации* [13] в зависимости от возможностей конечной системы и требований приложения.

- **VulkanGraphicsPipeline**: конвейер растеризации геометрических примитивов с использованием Z-буффера, который включает в себя настройку полностью программируемых этапов, таких как вершинные и фрагментные шейдеры, и частично конфигурируемых — сборка примитивов, растеризация, тест глубины и трафарета, а также альфа-смешивание. В отличие от графического API OpenGL, который позволяет динамически изменять процесс растеризации без явного создания объектов конфигурации, API Vulkan требует мануального создания полностью сконфигурированного неизменяемого графического конвейера, который позже может быть использован для отображения примитивов.

6.3. Система визуализации

Для доступа к графической функциональности высокого уровня был реализован класс **RenderEngine**, который поддерживает список видимых объектов и источников света на сцене, а также осуществляет отображения этих объектов в окно операционной системы или внеэкранный буфер.

Видимые объекты сцены реализуют интерфейс **IRenderable**, что позволяет им реагировать на события, генерируемые RenderEngine, и отображать себя независимо от структуры сцены или окружения. В ка-

честве стандартной реализации представлен **RenderableMesh**, который позволяет отображать геометрию, состоящую из индексированных вершин с фиксированным числом атрибутов.

Для конфигурации отображения реализован **Material**, который хранит состояние конвейера растеризации, а также поддерживает произвольные параметры отображения, которые могут быть установлены посредством типобезопасного интерфейса, автоматически упакованы и скопированы в видеопамять для последующего использования при отображении на GPU. Материалы поддерживают раздельное владение ресурсами GPU, что позволяет сократить расход видеопамяти и уменьшить число изменений состояния GPU за счет предварительно упорядочивания отображаемых объектов в зависимости от их материалов.

В качестве стандартных алгоритмов освещения доступны:

- Затенение объекта выбранным фиксированным цветом
- Затенение объекта с учетом интенсивности освещения сцены на основе модели Ламберта с использованием теневых карт для создания теней
- Физически-корректное затенение с учетом интенсивности освещения на основе вариации модели Кука-Торренса, которая была предложена в работе Брайана Кариса [8] как одна из наиболее визуально приемлемых моделей для вычисления в режиме реального времени. На рис. 3 представлен пример отображения геометрии с использованием данного алгоритма затенения.

Объекты, реализующие интерфейс **IPostEffect**, позволяют последовательно применять экранные эффекты к сформированному в процессе растеризации изображению. Для каждого эффекта доступна информация о цвете точки и ее глубине в соответствующем Z-буфере. В качестве стандартных эффектов реализованы черно-белый фильтр и фильтр инвертирования цвета, пример применения которых представлен на рис. 4.

Рис. 3: Физически-корректное затенение 3D-геометрии с использованием вариации модели Кука-Торренса

Рис. 4: Применение экранных эффектов к растеризованному изображению

(a) Черно-белый фильтр

(b) Фильтр инвертирования цвета

Заключение

В ходе выполнения работы получены следующие результаты:

- Проведено исследование предметной области
- Проведено сравнение существующих решений
- Разработаны и формализованы требования к проекту
- Реализованы компоненты библиотеки в соответствии с заявленными требованиями
- Реализована поддержка Vulkan API

Реализованная графическая библиотека была успешно *интегрирована* в проект CoreCVS студентом 371 группы Сультимом Цырендашиевым в рамках его курсовой работы. В качестве основы для интеграции использовалась платформа Qt виджетов для создания поверхности отображения.

Несмотря на скромную функциональность реализованной графической библиотеки, ее ядро в сочетании с поддержкой Vulkan API будет использовано в дальнейшем для расширения и создания 3D визуализации уже в рамках проекта CoreCVS.

Список литературы

[1] Akenine-Mller Tomas, Haines Eric, Hoffman Naty. Real-Time Rendering, Fourth Edition. — 4th edition. — USA : A. K. Peters, Ltd., 2018. — ISBN: 0134997832.

[2] BSF Framework // github. — Access mode: `https://github.com/GameFoundry/bsf` (online; accessed: 14.11.2019).

[3] Computer Vision Primitives Library // github. — Access mode: `https://github.com/PimenovAlexander/corecvs` (online; accessed: 11.11.2019).

[4] Direct3D 12 Graphics // Microsoft Online Documents. — 2018. — Access mode: `https://docs.microsoft.com/ru-ru/windows/win32/direct3d12/direct3d-12-graphics?redirectedfrom=MSDN` (online; accessed: 11.12.2019).

[5] Filament Graphics Library // github. — Access mode: `https://github.com/google/filament` (online; accessed: 16.11.2019).

[6] Group The Khronos Working. OpenGL Shading Language 4.4 Specification // Khronos Registry. — 2016. — Access mode: `https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.4.40.pdf` (online; accessed: 20.11.2019).

[7] High Level Shading Language for DirectX // Microsoft Online Documents. — 2018. — Access mode: `https://docs.microsoft.com/en-us/windows/win32/direct3dhlsl/dx-graphics-hlsl` (online; accessed: 20.11.2019).

[8] Real Shading in Unreal Engine 4 : Rep. / Epic Games ; Executor: Brian Karis : 2013. — Access mode: `http://blog.selfshadow.com/publications/s2013-shading-course/karis/s2013_pbs_epic_notes_v2.pdf` (online; accessed: 02.05.2020).

[9] Metal Programming Guide // Apple Documentation Archive. — 2018. — Access mode: https://developer.apple.com/library/archive/documentation/Miscellaneous/Conceptual/MetalProgrammingGuide/Introduction/Introduction.html (online; accessed: 12.12.2019).

[10] Ogre3D Engine // github. — Access mode: https://github.com/OGRECave/ogre (online; accessed: 15.11.2019).

[11] The Khronos Working Group. OpenGL 4.4 Specification // Khronos Registry. — 2014. — Access mode: https://www.khronos.org/registry/OpenGL/specs/gl/glspec44.core.pdf (online; accessed: 9.12.2019).

[12] The Khronos Working Group. SPIR-V 1.5 Specification // Khronos Registry. — 2018. — Access mode: https://www.khronos.org/registry/spir-v/specs/unified1/SPIRV.pdf (online; accessed: 20.11.2019).

[13] The Khronos Working Group. Vulkan 1.1 API Specification // Khronos Registry. — 2019. — Access mode: https://www.khronos.org/registry/vulkan/specs/1.1/html/vkspec.html (online; accessed: 10.11.2019).

[14] Wikipedia. Cel Shading // Wikipedia, the free encyclopedia. — Access mode: https://en.wikipedia.org/wiki/Cel_shading (online; accessed: 9.12.2019).