

**FIT DEPARTMENT**

Computer Science and System Design

Software and Hardware Development



**Software and Hardware Development**

**System Requirements Specification for project #6: “Rendering”**

Version: Cuda6

Date: 13.11.2019

Authors: Yaroslav Odintsov, Artem Plusnin, Ekaterina Dubinskaya.

**Version control page**

|  |  |  |  |
| --- | --- | --- | --- |
| **0.5** | **31.10** | **Use-cases;** **System-wide functional requirements; Extensibility.** | **5, 6,**  **7** |
| **0.5** | **9.11** | **Introduction; Actors; Use-cases; System-wide functional requirements.** | **4, 5,**  **6, 7,**  **8** |
| **Cuda6** | **11.11** | **New version information updated: System-wide functional requirements.** | **7, 8** |
| **Cuda6** | **13.11** | **New version testing results: Performance.** | **9** |
| **Cuda6** | **18.11** | **Use-cases Editing and visualization. System-Wide functional requirements.** | **7, 8,**  **9, 10,**  **11** |
| **Cuda6** | **22.11** | **System-Wide functional requirements.** | **11** |
| **Cuda6** | **1.12** | **System-Wide functional requirements.** |  |
|  |  |  |  |

**Table of contents:**

1. [Introduction](#Introduction)

2. [Glossary](#Glossary)

3. [Actors](#Actors)

4. [Functional requirements](#FunctionalRequirements)

4.1. Menu Use-cases

4.1.1. Use-case “Create scene”

4.1.2. Use-case “Scene import”

4.1.2. Use-case “Scene export”

4.2. System Use-cases

4.2.1. Use-case “Render”

5. [System-wide functional requirements](#SystemWideFunctionalRequirements)

6. [Non-functional requirements](#NonFunctionalRequirements)

6.1. Environment

6.2. Performance

6.3. Reliability

6.4. Extensibility

**Introduction**

Our team are developing application that works with 3D graphics. Main goal is to make render that uses ray tracing method. Ray tracing allows for dramatically more lifelike shadows and reflections, along with much-improved translucence and scattering. The algorithm considers where the light hits and calculates the interaction and interplay much like the human eye would process real light, shadows, and reflections, for example. The application itself includes some other functionality that will help user to explore and use ray tracing rendering. It makes a picture from 3D model. This model can be imported, built or edited in stock editor.

**Glossary**

“UI” – User interface.

“RT” – Ray Tracing

“Qt” – WinApi minimized.

“Starting scene” – the scene that run by default.

“K-D Tree” – is a multidimension binary search tree.

“Pixmap” – is an image format.

**Actors**

USER – person who uses application to render the models.

RTS – application itself.

**Functional requirements**

In-Menu use-cases:

Use-case ”Create Scene”:

**Actors:** *User.*

**Goals:** *create new empty scene.*

**Precondition:** *Open Menu.*

**Trigger condition:** *No such.*

**Extensions:** *No such.*

**Mains success scenario:**

1) USER presses “Create scene” button

2) RTS checks if there are any unsaved files opened

3) RTS generates empty scene

4) RTS visualize empty scene

**Alternative scenario “Unsaved files”:**

When point 2 main success scenario fails

1) RTS writes the USER warning message with “Save” and “Don’t Save” buttons

2) USER presses “Save”

3) RTS run “Scene save” use case from point 2

4) RTS generates empty scene

5) RTS visualize empty scene

**Alternative scenario “Unsaved files 2”:**

When USER don’t want to save in point 1 of alternative scenario “Unsaved files”

1) USER presses “Don’t Save”

2) RTS run “Scene save” use case from point 2

3) RTS generates empty scene

4) RTS visualizes empty scene

**Notes:** Empty scene has no objects on it.

Use-case ”Scene Import”:

**Actors:** *User.*

**Goals:** *import a new model onto the scene.*

**Precondition:** *Open Menu.*

**Trigger condition:** *Chosen file does not exist.*

**Extensions:** *No such.*

**Mains success scenario:**

1) USER presses “Import scene” button

2) RTS opens file explorer

3) USER chooses file from disk

4) USER presses “Import” button

5) RTS checks if file format computable or not

6) RTS parses the file

7) RTS builds scene

8) RTS visualizes scene

**Alternative scenario “Incomputable file format”:**

When point 5 main success scenario or point 5 alternative scenario fails

1) RTS writes the USER warning message

2) RTS opens file explorer

3) USER chooses file from disk

4) USER presses “Import” button

4) RTS checks if file format computable or not

5) RTS parses the file

6) RTS builds scene

7) RTS visualizes scene

**Notes:** File can be chosen only in supported formats.

Use-case ”Scene Save”:

**Actors:** *User.*

**Goals:** *save scene on disk.*

**Precondition:** *Open Menu.*

**Trigger condition:** *No such. (Windows can trigger “Not enough space” or “Can’t write in directory”)*

**Extensions:** *No such.*

**Mains success scenario:**

1) USER pressed “Save scene” button

2) RTS opens file explorer

3) USER chooses place on disk

4) USER types file name in the text box

5) User chooses file format from the list of formats

6) USER press “save” button

7) RTS saves file to the chosen place.

**Alternative scenario “Windows triggered saving”:**

When Windows triggered point 7 main success scenario or point 7 alternative scenario

1) RTS writes the USER warning message

2) RTS opens file explorer

3) USER chooses place on disk

4) USER types file name in the text box

5) User chooses file format from the list of formats

6) USER press “save” button

7) RTS saves file to the chosen place.

**Notes:** Points 3, 4, 5 can be done in any other order; File format list consists of only supported formats and includes RTS own format ”.fl3k$$” that is the best one if USER want to continue working with this scene using RTS later.

Editor use-cases:

Use-case ”Add Object”:

**Actors:** *User.*

**Goals:** *create new object on scene.*

**Precondition:** *No such.*

**Trigger condition:** *Bad settings input.*

**Extensions:** *No such.*

**Mains success scenario:**

1) USER chooses object shape

2) USER presses “Add” button

3) RTS opens object settings block

4) USER chooses object settings:

-position

-rotation

-shininess

-diffuse color

-emissive color

-specular color

-ambient intensity

-transparency

5) RTS saves object settings to scene data

6) RTS visualizes scene with new settings

**Alternative scenario “Incomputable settings”:**

When point 4 main success scenario or point 2 alternative scenario “incomputable settings” has incomputable settings

1) RTS writes the USER warning message with information about incomputable object settings

2) USER chooses object settings:

-position

-rotation

-shininess

-diffuse color

-emissive color

-specular color

-ambient intensity

-transparency

3) RTS saves object settings to scene

4) RTS visualizes scene with new settings

**Notes:** incomputable settings example is invalid values.

Use-case ”Edit Object”:

**Actors:** *User.*

**Goals:** *edit object settings on scene.*

**Precondition:** *add object or import scene with objects.*

**Trigger condition:** *Bad settings input.*

**Extensions:** *No such.*

**Mains success scenario:**

1) USER chooses object on scene

2) USER presses “Edit” button

3) RTS opens object settings block

4) USER chooses object settings:

-position

-rotation

-shininess

-diffuse color

-emissive color

-specular color

-ambient intensity

-transparency

5) RTS saves object settings to scene data

6) RTS visualizes scene with new settings

**Alternative scenario “Incomputable settings”**

**Notes:** incomputable settings example is invalid values.

Use-case ”Delete Object”:

**Actors:** *User.*

**Goals:** *delete object on scene.*

**Precondition:** *add object or import scene with objects.*

**Trigger condition:** *No such.*

**Extensions:** *No such.*

**Mains success scenario:**

1) USER chooses object on scene

2) USER presses “Delete” button

3) RTS deletes information about object from scene data

4) RTS visualizes scene with new settings

**Notes:**.

Use-case ”Visualization”:

**Actors:** *RTS.*

**Goals:** *represent the scene data to screen.*

**Precondition:** *Open Application.*

**Trigger condition:** *No such.*

**Extensions:** *No such.*

**Mains success scenario:**

1) RTS calls visualization

2) RTS builds pixmap due to the scene data

3) RTS represents pixmap on screen

**Notes:** *It’s just simple representation it does not include light behavior.*

Use-case ”Render”:

**Actors:** *User.*

**Goals:** *Get scene rendered.*

**Precondition:** *Open Application.*

**Trigger condition:** *No such.*

**Extensions:** *No such.*

**Mains success scenario:**

1) USER types ray tracing depth

2) USER chooses a render type

3) USER presses “Render” button

4) RTS sends package with all information that USER choose

5) RTS blocks scene

6) RTS starts rendering algorithm

**Notes:** *All functionality depends on user’s decision.*

**System-wide functional requirements**

Light: Algorithm simulates the nature properties: a light source emits a ray of light which travels, eventually, to a surface that interrupts its progress.

Ray: Ray will be a straight line (ignoring relativistic effects). Any combination of four things might happen with this light ray: absorption, reflection, refraction and fluorescence.

Absorption: A surface may absorb part of the light ray, resulting in a loss of intensity of the reflected and/or refracted light.

Reflection: It might also reflect all or part of the light ray, in one or more directions.

Refraction: If the surface has any transparent or translucent properties, it refracts a portion of the light beam into itself in a different direction while absorbing some (or all) of the spectrum (and possibly altering the color).

Algorithm: When a ray hits a surface, it can generate up to three new types of rays: reflection, refraction, and shadow. A reflection ray is traced in the mirror-reflection direction. The closest object it intersects is what will be seen in the reflection. Refraction rays traveling through transparent material work similarly, with the addition that a refractive ray could be entering or exiting a material. A shadow ray is traced toward each light. If any opaque object is found between the surface and the light, the surface is in shadow and the light does not illuminate it. This recursive ray tracing added more realism to ray traced images.  
Camera:

**Non-functional requirements**

Environment

Users’ system requirements (minimal):

- Personal computer with Windows.

- Processor: any.

- Graphic adapter: any.

- X Mb of free space on a disc.

Users’ system requirements (recommended):

- Personal computer with Windows 10.

- Processor: Intel Pentium G4560/Amd Ryzen 3 1200.

- Graphic adapter: (NOT AMD!) nVidia GTX1050Ti.

- X Mb of free space on a disc.

Performance

Testing results on recommended stand with version “Cuda6”:

2.000+ seconds scene rendering without tree (125 objects).

0.900 seconds scene rendering with tree (125 objects).

6.128 seconds write to file.

x.xxx seconds import/export scene.

x.xxx second building application interface.

x.xxx seconds to drawing.

Program starts in x.xxx seconds.

Reliability

Tests soon.

Extensibility

Black-Box. In black-box extensibility no details about a system’s implementation are used for implementing deployments or extensions; only interface specifications are provided. This type of approach is more limited than the various white-box approaches. Black-box extensions are typically achieved through system configuration applications or the use of application-specific scripting languages by defining components interfaces.