Surgery Scene Representation in 3D Simulation Training SDK

Anton Ivaschenko, Nikolay Gorbachenko
Samara State Aerospace University
Samara, Russia
{anton.ivashenko@, eaglegor}@gmail.com

Alexandr Kolsanov
Samara State Medical University
Samara, Russia
avkolsanov@mail.ru

Andrey Kuzmin
Penza State University
Penza, Russia
flickerlight@inbox.ru

***Abstract*—This paper introduces a software development kit (SDK) to provide IT developers and medical community a platform to build new simulation technologies for surgery training. There is described a formal model for simulation objects, scenes and scenarios representation and a software solution for objects interaction modeling and surgery scene simulating. The results are illustrated by the simulation suite for laparoscopy surgery training delivered and installed at Samara State Medical University.**

I. Introduction

Simulation training is actively employed at medical universities nowadays. Particularly surgery training has become a promising area of full range of simulation technologies application including 3D visualization, motion capture, knowledge representation, etc. In this area, there is a strong market request for new information technologies and their applications that will be able to provide an efficient training of surgeons considering their specialization and difference in educational programs at medical universities and training centers.

First results in this area were achieved under the "Virtual Surgeon" and "Inbody Anatomy" projects that were carried out in 2012 – 2014 and include a number of simulation training suites for endovascular and laparoscopy surgery [1 – 2]. These products were successfully probated in several medical universities in Russia. During the process of their deployment and technical support there was gathered a remarkable feedback from professors and IT departments.

There was identified a strong request to separate the training suites into a number of autonomous components that can be used by universities themselves to build own solutions and adapt them for specific educational programs.

This feedback led to motivation of a new research project started in 2014 and aimed to develop a number of software components for surgery training, an extensive set of 3D models of human body, and special software used by universities to develop their own products capable to provide personalized simulation and training. In addition to it there was designed an Internet platform that provides to its users an opportunity to exchange and share new simulation technologies and products in integrated information space.

To meet the goals there has been developed a specialized software development kit (SDK) [3], which contains a number of components that can be used to implement a large variety of simulation solutions for surgery training. In this paper there are described some technical solutions for operating scene ontology model formalization and its implementation in the proposed SDK.

II. State of the Art

Intensive application of simulation tools for surgery training at medical universities and specialized training centers requires implementation of the most up-to-date technologies in robotics, 3D modeling, electronics and software engineering. Still the combination of the most realistic visualization of surgery field and adequate haptical feedback of manipulators that simulate surgery instruments can appear to be not enough for effective educational process [4 – 6]. At the same time, some simple versions of training suites that are comparatively cheap and easy in use can give a significant improvement to a surgery training technique. Consequently it is reasonable to introduce a specialized software development kit (SDK) aimed for different developers and medical community to provide various solutions for surgery simulation training.

Technical aspect can be characterized by an extensive use of emerging and innovative technologies to provide high realism of visual scene and force feedback. Among the most widespread areas of simulation for surgery training, there are laparoscopy, laparotomy and endovascular diagnostics and surgery [7, 8]. The mentioned above solutions are based on visualization of 3D scenes that represent surgery fields for different cases and simulation of surgery intervention by means of specifically designed manipulators. The concept is pretty close to gaming simulation: the student has a certain situation described by visual model with predefined features and can perform a number of actions getting the response that simulates the real human body behavior.

Due to high complexity and uncertainty specific for real surgery intervention and uniqueness of every individual surgery scene the simulation is often simplified: the number of cases is limited to typical ones (normal and pathologic), and the possible behavior of the model is captured by a certain scenario. This helps implementing the simulation of not all the organs of a human body with realistic physiology and

feedback, but only a fragment relevant to the current surgery case. Therefore, the student can perform a limited number of actions at a certain moment of time; otherwise, the system will terminate the game with an exception.

Besides there can be extracted two types of simulators both popular in the market: without force feedback [9] (SIMENDO laparoscopy from SIMENDO and LAP-X from Epona Medical), and providing force feedback (CAE LaparoscopyVR Simulator from CAE Healthcare (http://caehealthcare.com/), and LapSim Haptic System from Surgical Science Sweden AB (http://www.surgical-science.com/). Simbionix provides LAP Mentor training suite with both options, either including or excluding force feedback.

It should be mentioned that each type of simulators has its own niche. Training suites without force feedback are constructively simple and relatively cheap. They allow surgeons to gain basic surgery skills and can help to evaluate a theoretical qualification and career potential of a medical student who is going to perform certain surgery interventions. Those that provide force feedback are able to offer a new range of educational techniques and bring the student closer to real life practice. One of the main problems here is to train the students to operate surgical instruments, which requires studying uncommon hand motions.

In the area of 3D modeling there can be identified a number of software development projects that become frequently used in visual simulators. In 2006 the generic approach of modeling internal organs was developed as a part of the GiPSi (General Physical Simulation Interface) project [10]. GiPSi is an open source/open architecture framework for developing organ level surgical simulations.

In 2007 the basic principles of the SOFA project were introduced [11]. This project is focused on the creation of the instrument set for the development and the comparison of different physical models of internal organs. The architecture allows combination of different physical properties and equation solvers introduced for this purpose. Extensive usage of third-party libraries based on SOFA allowed to boost simulations creation speed.

The described above state of the art analysis supported by the consultancy of medical professors and doctors from a number of leading medical universities in Russia results in a conclusion that in the area of surgery education there remains a variety of methodologies and educational technologies. From IT service point of view, this gives reasons for a relevance of SDK development used as a platform for new simulation solutions. This is a challenging technical problem: SDK should provide interoperability, compatibility and usability. To provide these features there was developed an architecture, were the data (3D models, scenes and scenarios) are separated from UI and core components.

III. SDK ONTOLOGY MODEL

Let us consider a surgical operation as a set of objects in a surgical field and a set of events occurring during the operation. We do not explicitly distinguish instruments and organs because they can be described using the same concepts. There are denoted the combination of these two sets by the scene Q:

$$Q = \{O, E\}, \quad (1)$$

where O is the set of scene objects and E is the set of events occurring during the surgical operation.

Event is described by its type and the set of attributes describing the particular situation:

$$E = \{e_i, i = 1...I\},\ e_i = \{e_{i,l}, l = 1...L_i\} \quad (2)$$

Scene object is described by its behavior aspects. Each behavior aspect is described by the two components. The first one is the set of states of the object having this behavior aspect. The second one is the set of transitions between these states. We denote the set of states of the scene object $o_k \in O$ by s_k. The single state of the scene object is denoted by $s_{kt} \in s_k$ as described by the set of attributes:

$$S = \{s_k, k = 1...K\},\ s_k = \{s_{k,t}, t = 1...T_k\},\ s_{k,t} = \{s_{k,t,j}, j = 1...J_{k,t}\} \quad (3)$$

The transition between the states $s_{k,t}$ and $s_{k,t'}$ of the object o_k after the event e_i has occurred is denoted by $r_{s_{kt}, s_{kt'}, e_i}$. Let us define this transition as a mapping of the state $s_{k,t}$ and the event e_i to the state $s_{k,t'}$:

$$r_{s_{kt}, s_{kt'}, e_i} : \{s_{k,t}, e_i\} \to s_{k,t'}. \quad (4)$$

The denotation $r_{k,t,t',i} = r_{s_{kt}, s_{kt'}, e_i}$ is introduced to make the equations look clearer. This denotation states that if the event e_i occurs and the scene object o_k is in the state $s_{k,t}$ then the scene object will move to the state $s_{k,t'}$.

Transition between the two states of the scene object may occur after several different events. We denote all possible transitions between the states $s_{k,t}$ and $s_{k,t'}$ by

$$r_{s_{kt}, s_{kt'}} = r_{k,t,t'} :\ r_{s_{kt}, s_{kt'}} = r_{k,t,t'} = \{r_{k,t,t'f}, f = 1...F_{k,t,t'}\} \quad (5)$$

The set of all possible transitions between all possible states of the scene object can be considered as a mapping of the set of all object states and the set of all event types to the set of all object states:

$$r_k = \{r_{s_{km}, s_{kn}}, s_{k,m} \in s_k, s_{k,n} \in s_k\},\ r_k : \{s_k, E\} \to \{s_k\} \quad (6)$$

Each single behavior aspect of the scene object is called the component. It is defined as follows:

$$c_{k,d} = \{\hat{s}_{k,d} \in s_k, \hat{r}_{k,d} \in r_k\},\ c_k = \{c_{k,d}, d = 1...D_k\}, \quad (7)$$

where c_k is a set of all components attached to the scene object o_k, $\hat{s}_{k,d}$ is the subset of object states containing only states affected by the component $c_{k,d}$, $\hat{r}_{k,d}$ is the subset of transitions between the states of o_k containing only the ones introduced by the component $c_{k,d}$.

For each pair of scene objects we define the set of interaction aspects. The single interaction aspect between the objects o_k and o_l is described by the set of events, the set of both objects states and the set of transitions between object states:

$$a_{k,l,h} : \{\overline{e_{k,l}} \subseteq E, \overline{r_{k,h}} \subseteq r_k, \overline{r_{l,h}} \subseteq r_l, \overline{s_{k,h}} \subseteq s_k, \overline{s_{l,h}} \subseteq s_l\}, \quad (8)$$

where $\overline{r_{k,h}}$, $\overline{r_{l,h}}$, $\overline{s_{k,h}}$, $\overline{s_{l,h}}$ are defined by the type of interaction aspect. $\overline{r_{k,h}}$ and $\overline{r_{l,h}}$ contain transitions between $\overline{s_{k,h}}$ and $\overline{s_{l,h}}$ respectively. The events subset $\overline{e_{k,l}}$ is defined by the mapping $\phi_{k,l}$:

$$\phi_{k,l} : \{c_k, c_l\} \to \{\overline{e_{k,l}}\}. \quad (9)$$

Interaction of each two scene objects is defined by the set of all their interaction aspects:

$$a_{k,l} : \{a_{k,l,h}, h = 1...H_{k,l}\}. \quad (10)$$

Any possible object behavior and interaction can be described using the approach we introduced. But our goal is not only to describe the behaviors and interactions on the scene but to reach the flexibility and the ability to adopt to changing requirements. To reach this goal we need to introduce an approach of combining objects behaviors and interactions.

Generally, the objects itself do not have any behaviors initially. During the development of the surgical case model the operator may extend the set of behaviors and interactions to reach the simulation goals.

Using the introduced models such extension is the association of a new component $c'_{k,i}$ to the object o_k:

$$(c_k + c'_{k,i}) : \{c_k, c'_{k,i}\} \to \tilde{c}_{k,i}, \tilde{c}_{k,i} = c_k \cup c'_{k,i} \quad (11)$$

As a result of this attachment the set of object states and the set of transitions between object states are affected:

$$s_k \to \tilde{s}_{k,i}, \; r_k \to \tilde{r}_{k,i} \quad (12)$$

It is natural to think that different components may affect the same object states (e.g. the organ may be damaged by the electrocoagulation or by the puncture but both events translate the organ to the "damaged" state). On the other hand, different components may affect absolutely different states (e.g. when the organ is damaged and grasped by the instrument, "damaged" and "grasped" are completely different by their nature). Considering both cases we define $\tilde{s}_{k,i}$ as a cartesian product of all states the object already has and the new states introduced by the component $c'_{k,i}$:

$$\begin{aligned} \tilde{s}_{k,i} &= s_k \times \hat{s}_{k,i} = \\ &= \{s_{k_1}\hat{s}_{k,i_1}, s_{k_2}\hat{s}_{k,i_1}, ..., s_{k_{T_k}}\hat{s}_{k,i_2}, s_{k_1}\hat{s}_{k,i_2}, ..., s_{k_{T_k}}\hat{s}_{k,i_{\hat{T}_{ki}}}\} = \\ &= \{\tilde{s}_{k,i_1}, \tilde{s}_{k,i_2}, ..., \tilde{s}_{k,i_{\tilde{T}_k}}\} \end{aligned}, \quad (13)$$

where $\hat{s}_{k,i} = \{\hat{s}_{k,i_t}, t = 1...\hat{T}_{ki}\}$ is the subset of states introduced by the component $c'_{k,i}$.

The set of transitions between object states must be updated in a more complex way to define transitions on the new combined set of object states.

When attaching a new component, we define the new behavior aspects of an object. Our model is event-based, therefore to make the object move over the states we must pass events to this object. As mentioned before, we use the mapping $\phi_{k,l}$ to define the events generated for each pair of objects, s owe need to change this mapping to define the new interaction on the scene.

Let us introduce the operation of linking the objects o_k and o_l through the subset of events $e' \subseteq E$:

$$\begin{aligned} &\phi_{k,l} \to \tilde{\phi}_{k,l}, \; (\phi_{k,l} \vee e') : \{\phi_{k,l}, e'\} \to \tilde{\phi}_{k,l}, \\ &\tilde{\phi}_{k,l} = \{c_k, c_l\} \to \hat{e}_{k,l} \cap e', \end{aligned} \quad (14)$$

where $\hat{e}_{k,l}$ is the set of events defined for the specified combination of components before linking and e' may contain any possible event types on the scene.

Let us consider the example. We describe two objects – "the dissector" (o_1) and "the liver" (o_2). We attach the component "grasp algorithm controller" to the "dissector" and the component "grasp algorithm" to the liver. Doing this we add the "may grasp" property to the "dissector" and the "may be grasped" property to the liver. But doing only this we cannot grasp the liver with the dissector because $\phi_{1,2} = \{c_1, c_2\} \to \varnothing$.

We have transitions between objects states defined but we have no events passing to the objects. This may be the expected result (in case of different difficulty levels) or the mistake of scene creator. Performing objects linking we update the mapping so that $\tilde{\phi}_{1,2} = \{c_1, c_2\} \to \{e_1, e_2\}$, where e_1 is the "objects collided" event and e_2 is the "grasp started" event.

After this operation is completed the environment has the information about event types to be generated between these objects and the corresponding interaction aspects may be defined (in our case it is the aspect called "grasping the liver with the dissector").

The two introduced operations (attaching new component and objects linking) allows us to create complex scenes based on more simple scenes. To adopt the models, we will sometimes need to do the opposite – make simpler scene base on more complex scene.

To make this possible we introduce the inverse operations – removing a component and unlinking objects:

$$(c_k + c'_{k,i}) - c'_{k,i} = c_k, \; (\phi_{k,l} \vee e') \setminus e' = \phi_{k,l} \quad (15)$$

Using the introduced approach to the surgical case modeling we can flexibly adjust the behavior of scene objects and create the wide range of scenes using the same building blocks.

IV. SOLUTION VISION

Development of surgery simulators for the purposes of training of students at medical higher schools is the actively developed research area today. There is a need for cheap and reusable training environment helping students to get the needed skills they will use in real operations. Virtual surgery simulators can become such an environment, because the simulations can be run many times and can model the wide range of surgical cases.

There are some commercial simulators (LapVR, Simbionix, LapSim) but they are expensive and have the closed architecture. Because of this fact, there are difficulties when the independent expert wants to extend the functions of simulator or adopt some surgical cases for his or her own purposes. To make such extension and adoption possible we need to develop an instrument set allowing developing new surgery simulators using minimal amount of time and effort and having very basic technical skills.

The first requirement (minimization of time and efforts) allows the medical experts to quickly adopt the simulations to the modern surgery approaches. This requirement could be met if the developed content and modules were highly reused while developing new operation models. The second requirement extends the range of content creators to the non-programmers and helps to build the rich content base. This requirement could be met if the simple way of scene description was introduced (WISIWYG-editors, script languages, etc.)

Let us focus on flexible and extendable approach of surgery simulations creation, which would be able to meet these requirements. The proposed SDK architecture for surgery simulations platform is based on the "Entity-component" design pattern. All scene objects described in the surgical case (instruments, organs, environment) are considered as simple containers without their own behavior. During the modeling the behavior aspects called components are associated with these scene objects. Each component is characterized by its name, type and a set of the interaction interfaces.

Components lifecycle control is the responsibility of a number of subsystems. Each subsystem is serving the group of components it owns. An example of such subsystem is the scene of physics engine controlling components modeling rigid body and soft body dynamic properties of the objects (see Fig. 1).

Subsystems are registered at the platform core when the plugins containing these subsystems are loaded. As a result, new behavior aspects become available to be attached to the scene objects. The logic is divided into the base logic (memory management, resource management, components creation and removal, logging etc.) implemented in the core and the specific logic (render algorithms, physics algorithms, hardware controllers etc.) entirely implemented in the loadable plugins. There may be several interchangeable implementations of the same plugin interface (e.g. physics plugin may be based on PhysX or Bullet having the same interface).

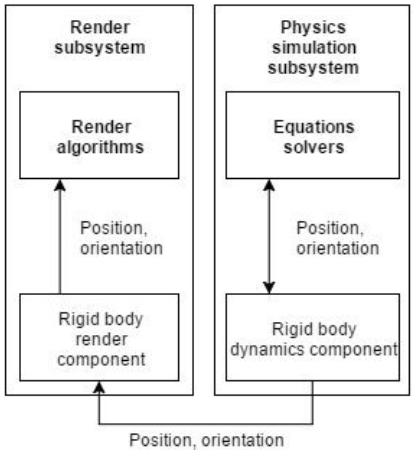

Fig. 1. Relations between the components and subsystems

Such approach has a set of advantages:

- there is no deep class hierarchy (less duplicated code when there are many combinations of behavior aspects);
- new behaviors may be introduced to the scene without recompilation of code (less time is consumed when developing a scene);
- content and algorithms may be distributed as loadable packages extending the functions of system without extensive changes to the existing system parts.

There are also some disadvantages of this approach:

- decomposition and isolation of program code inside the small components requires good programming skills (without this isolation the performance may decrease because of intensive communication between components);
- the approach is very generic and therefore it needs the good mechanisms of consistency control.

Considering decomposition of logic based on its specificity for the particular surgical case, we can build the hierarchy of program layers where the interaction mostly takes place only between the neighbor layers (see Fig. 2).

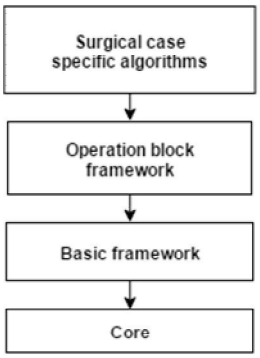

Fig. 2. Hierarchy of program layers

The core contains low-level generic services without any relation to the subject area. It is responsible for maintaining

the environment where the components have necessary resources and may effectively communicate to each other.

The basic framework contains the generic use services such as render subsystems, physics subsystems, scripting subsystems, GUI etc. There services are not related to the surgery simulations and represent the standard game engine subsystems.

The operation block framework contains the components specific for the operation class. For example, the endoscopic framework may contain grasping, cutting, puncture algorithms, body state control algorithms, hardware controllers etc. Surgical case specific algorithms contain the algorithms involved only in small subset of operation simulations. An example is the quality control of operator skills which is specific for every operation case.

On each level components and subsystems are defined using the generic interfaces which makes it possible to interchange different modules to reach particular simulation goals.

V. INSTRUMENT-ORGAN INTERACTION

Considering the high computational cost of surgery simulation algorithms, the algorithms of objects that simulate human body parts and objects that simulate instruments should be as fast as possible while still saving the flexibility of scene behaviors. Let us consider an example of endoscopic grasping algorithm. The endoscopic instrument is simulated as a compound rigid body containing moving parts. The organ is considered as a set of vertices with the constrained degrees of freedom. The constraints are based on the Hooke's law.

The simplified task of designing the grasping algorithm may be formulated as follows:

- We need to make the system respect the physical constraint: when the angle between the branches is smaller than the threshold, and there are vertices between branches, the position of this vertices in the local coordinate system of an instrument remains the same and independent from any instrument movements.
- We so not consider the haptics in this research, therefore we do not consider that the organ pushes the instrument away (the instrument is considered to have the infinite mass).
- We may divide our task into several steps:
    - detecting the potentially grasped vertices;
    - detecting the moment of grasping;
    - creation and respecting of described physical constraint;
    - detecting the moment when the vertices must be released;
    - destruction of physical constraint.

One of the most computationally complex part is detecting the potentially grasp vertices, because it needs the usage of complex collision detection algorithms for any instrument and organ on the scene. However, it is not necessary to make this check on every time step for every organ and instrument. We may perform this check only when the instrument is near the desired organ. This check (the distance to the organ) may be implemented in a fast way and spare the processor time. So we define the first component, trigger, responsible for this fast pre-check. We attach this component to the instrument.

The constraint itself needs to be included in the physics simulation process. We do not consider, how it is implemented exactly but we group all the constraint logic in a component called physical constraint. The component is automatically associated to the organ.

Different organs (human body parts) require various implementations of physical constraints. Moreover, there may be several organs of different nature in the same scene. Therefore, we need to assign the particular constraint implementation to the particular organs. To do so there is introduced the component called physical constraints factory, which is responsible for the creation of the correct instance of constraint. The component is also associated to the organ.

The last component we introduce is an algorithm controller responsible for tying the other three together. The interaction scheme is shown in Fig. 3.

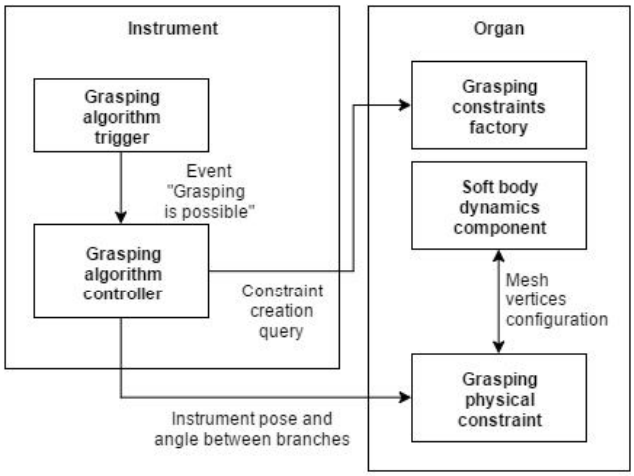

Fig. 3. Interaction scheme

The same approach is used to design other interaction algorithms (cutting, puncture, coagulation). The main principles are the following:

- getting fast algorithm start pre-check;
- different constraints implementations for different organs;
- minimization of communication between the components.

The resulting algorithm has simple dataflow, and its parts are easily interchangeable.

VI. Creation of a Surgical Case Modeling Scene

Generating of a new surgical case can be illustrated by the following example given for the endoscopic clipping scene. The surgical field for this example contains the elastic tube fixed on both ends, the camera and the endoscopic clip-applicator controlled by the operator (see Fig. 4). The task for the operator is to correctly place three clips on the tube.

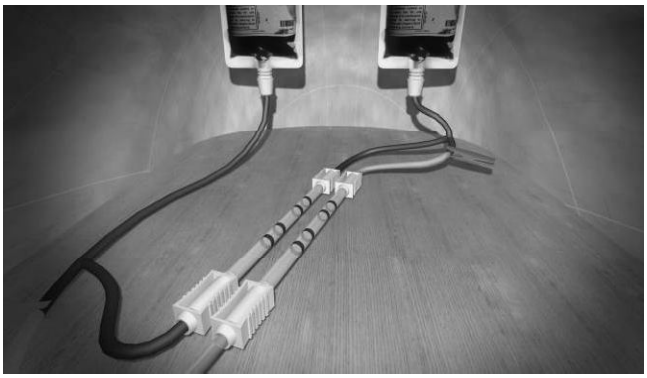

Fig. 4. Endoscopic clipping scene

There are identified 4 objects: the tube itself (model of a blood vessel), clip-applicator, endoscopic camera, operation script controller. These objects parts and interactions are shown in Fig. 5.

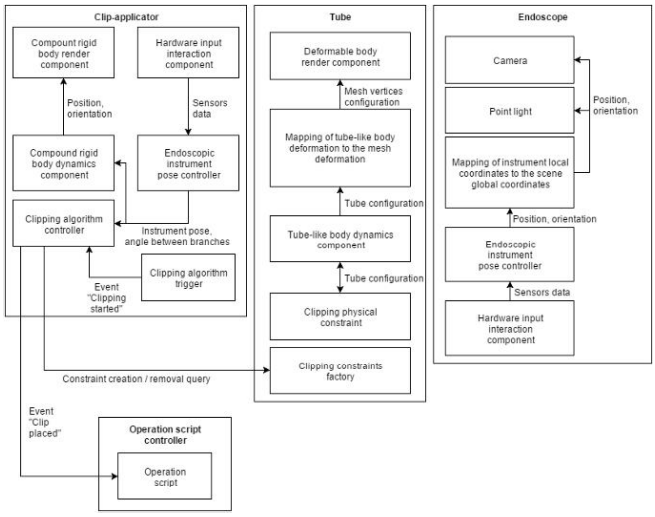

Fig. 5. Clipping scene objects

The most of the implemented components are reused in other scenes, e.g. the instrument pose controllers are used in any endoscopic operation, clipping algorithm are used in any operation case including blood vessels, render components may be used almost in any possible operation. New components need only to be developed when the simulation goal cannot be reached using already existing components.

The algorithm of scene creator's actions during the scene creation is shown in Fig. 6.

VII. Evaluation and Tests

Based on the proposed approach there was developed a solution using the following third-party libraries: OGRE 1.9 (www.ogre3d.org/), PhysX 3.3.3 (https://developer.nvidia.com/), CeGUI 0.8.4 (cegui.org.uk), SDL 2.0 (www.libsdl.org), AngelScript 2.30 (www.angelcode.com), Assimp 3.1.1 (assimp.sourceforge.net), Eigen 3.2.3 (eigen.tuxfamily.org), Boost 1.55 (www.boost.org).

In order to evaluate the proposed solution there were implemented two scenes. The parts and interactions between objects are shown in Fig. 7.

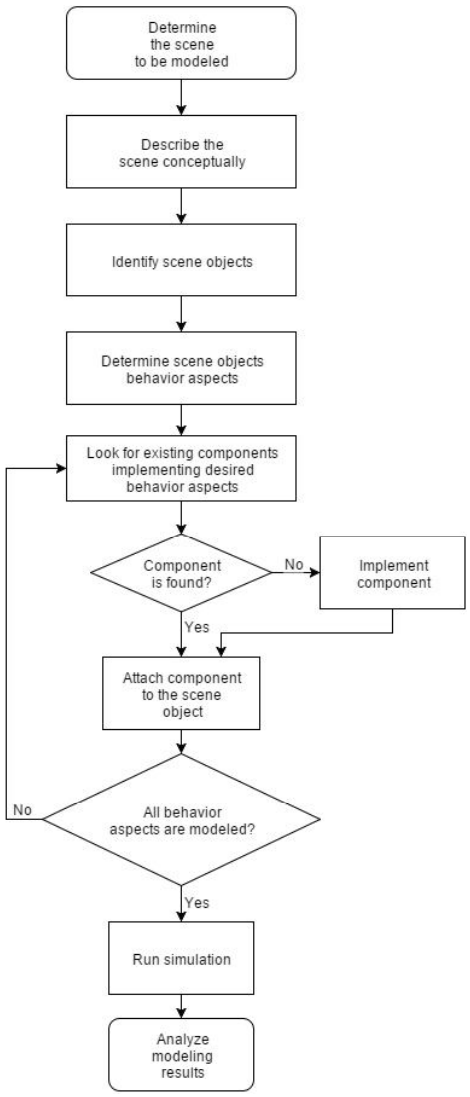

Fig. 6. Scene creating algorithm

The purpose of the first scene (see Fig. 8) is to check if the approach may be used in simple real scene modeling. The scene represents the simplified endoscopic camera positioning skills training module. It contains two active objects, several passive environment objects, endoscopic camera controlled by the operator and the graphical user interface (crosshair and hints). All behavior aspects (including GUI) are implemented as components.

The second scene is a benchmark. The proposed approach assumes using autonomous components to construct new scenes. It benefits in high adaptability and configurability, but leads to additional time expenditures due to several layers of indirection. Therefore, it is expected that implementing the proposed approach will result in lower performance comparing with the performance of static compilation approach when all the logic is encapsulated inside the scene object (no components and dynamic linking).

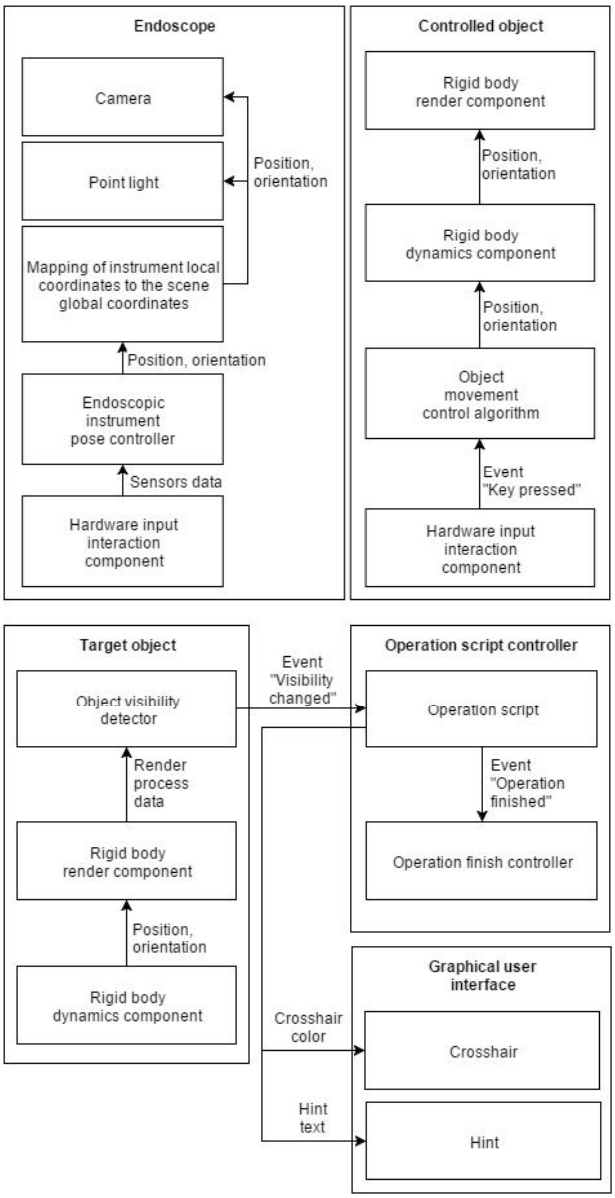

Fig. 7. Interaction between the objects in scene

To identify the value of time spending there was created a scene that consists of a continuously increasing set of similar objects (standard teapot model) rotating around their local y-axis. Every 200 frames the objects count increases (see Fig. 9).

There were implemented two variants of this scene – component-based and static compiled. Both versions use the same algorithms and perform the same actions during the frame.

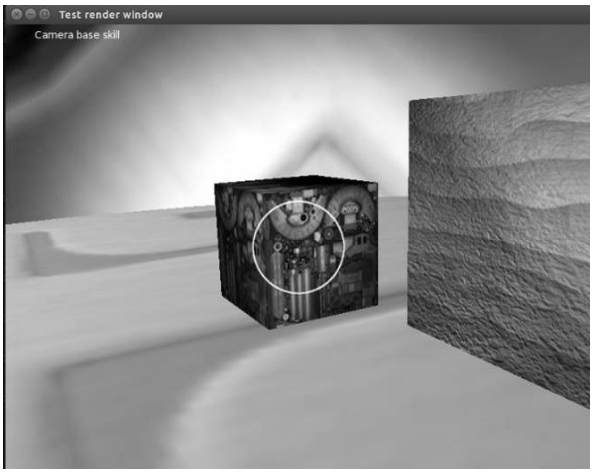

Fig. 8. Scene for endoscopic camera positioning skills training

The results of benchmarking study of these two approaches in terms of performance are given in Fig. 10. The performance metric is the average frame update time calculated for 100 frames (50 frames before object count increase and 50 frames after object count increase are not counted). Each experiment was repeated 10 times and the average values for all experiments were calculated.

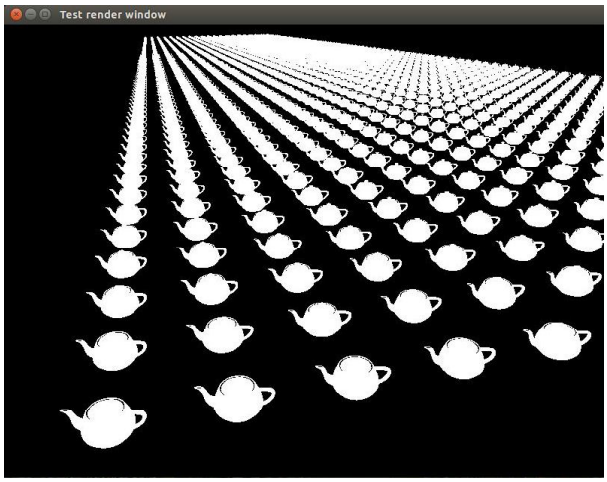

Fig. 9. Scene for benchmarking

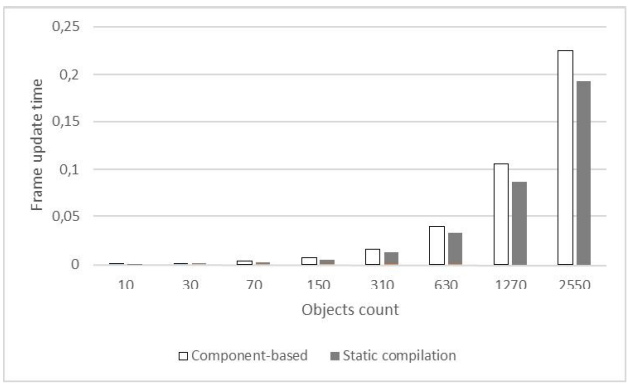

Fig. 10. Benchmarking

The experiment results show that the component-based approach is approximately 20% slower than the direct static compilation approach, which is acceptable in practical applications.

The results show that the proposed approach can be used to implement completely different behavior aspects of scene objects in a framework with the unified architecture. The slow down because of indirection layers is relatively small comparing with the naïve (static compilation) approach.

VIII. Implementation

The described above approach and SDK were used to develop a number of surgery training suites for laparoscopy and endovascular simulation. To provide realistic 3D scenes there was developed an original library of human body parts models that is currently available in the market as an independent product under the trademark “Inbody Anatomy”. By the moment of this paper presentation there were designed, developed and conjoined up to 3000 models of human body parts (see Fig. 11 – 12) combined to 12 layers of human body, including the ligaments, blood vascular system, innervations system, outflow tracts, lobar and segment structures of internals. To develop the models there were used real digital volume computer tomography and magnetic resonance tomography images.

These models were used to develop realistic scenes of surgery intervention (see Fig. 14). The models were fashioned with the help of specifically designed shaders and some fragments like jars, liquids and blood were implemented in software. These shaders and algorithms were implemented apart from the models; and any new models including those that are implemented by third party developers can be used to generate surgery scenes as well.

To provide highly realistic visual and physical models there were used PhysX (https://developer.nvidia.com/gameworks-physx-overview) and Bullet (http://bulletphysics.org) middleware. The inner parts of a human body are simulated in the scene by soft body models and surgery instruments are simulated by rigid bodies. The value of the feedback force is calculated on the basis of current geometrical position of soft and rigid bodies in the scene considering their deformation and/or topology distortion.

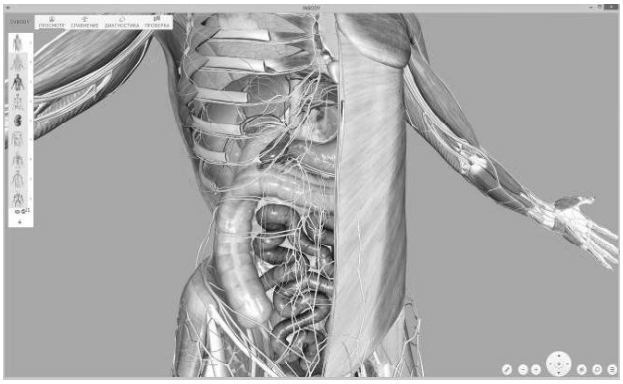

Fig. 11. Anatomy 3D models

Fig. 12. “Inbody Anatomy” product in use

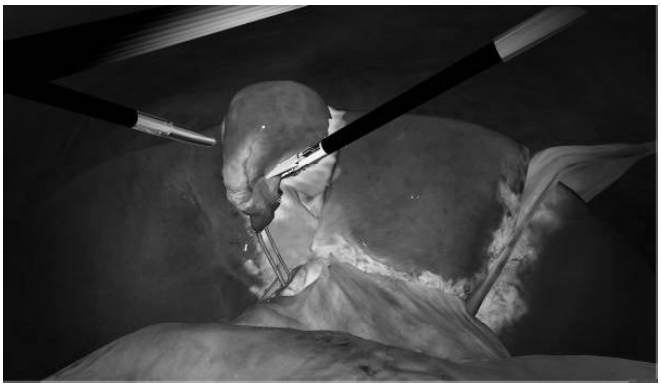

Fig. 13. 3D scene for laparoscopic cholecystectomy

The software was developed in C++ using .NET C# as a platform for user interface and infrastructure support. It provides realistic simulation of surgical intervention with the usage of different laparoscopic instruments and video camera – endoscope in 3D scenes. There were 3D models of the human body and operative intervention scenarios also developed that help the learning of basic skills and techniques of surgical treatment.

Hardware partials like laparoscopy manipulator, camera-endoscope etc. were also insulated and reorganized as autonomous components with specific API providing interoperability with other training suites modules. To provide realistic physical feedback there was designed a new construction of training suites and developed special software

to simulate the process of operative surgery. To simulate the movement of different surgical instruments an original construction was developed. The main assembly unit is the same for all types of manipulators as they differ only in mount attachment, orientation to console and types of adjustable portable handles.

The laparoscopic manipulator was built using 4 Dynamixel actuators (http://www.robotis.us/dynamixel/) that provide four degree-of-freedom feedback and movement. The main feature of the construction is that the entire outfit is integrated into the manipulator. This helps to reduce any transmission and as a result increases robustness and feedback sensitivity as compared with widely spread analogs. Optical sensors are used to capture the movement of camera-endoscope, and electronic actuators are introduced to simulate the movement of manipulators and provide force feedback. Such construction allows the introduction of different positions of manipulators and simulates cholecystectomy, hernioplasty and gynecology. Special scenes and study methods are provided to train basic surgical skills.

The developed laparoscopic training suite in use is presented at Fig. 15 and the simulation center capable to support surgery training is presented in Fig. 16. The center was established at Samara State Medical University and is currently launched to practical use.

Fig. 14. Laparoscopy training suite

IX. Conclusion

In this paper there is introduced a Software Development Kit (SDK) to provide IT developers a platform to build new simulation technologies for medicine. Special attention is given to surgery scene representation and unified implementation. The ontological model provides formalization of surgery scene objects interaction in different states. There is proposed a hierarchy of program layers and the framework that contains the generic use services such as render subsystems, physics subsystems, scripting subsystems, GUI etc. The operation block framework contains the components specific for the operation class. Surgical case specific algorithms contain the algorithms involved only in small subset of operation simulations.

Fig. 15. Simulation center at Samara State Medical University

The proposed approach allows the developers of new training suites to better specify the requirements, concretize the scope, prepare effective tests and improve training efficiency. High perspectives of 3D Surgery simulation software development kit application in medical simulation education prove its practical utility and motivate further developments in this area.

X. Acknoledgement

This research was financially supported by the Ministry of Education and Science of Russian Federation (grant 2014-14-579-0003), contract 14.607.21.0007.

References

[1] A. Ivaschenko, A. Dmitriev, A. Cherepanov, A. Vaisblat and A. Kolsanov, “Virtual Surgeon” training suite for laparoscopy, endovascular and open surgery simulation”. *Proceedings of ESM 2013*, Lancaster university, UK, EUROSIS-ETI, 2013, pp. 114 – 118

[2] A.V.Kolsanov, A.V. Ivaschenko, A.V. Kuzmin and A.S. Cherepanov, “Virtual Surgeon system for simulation in surgical training”. *Biomedical Engineering*, 2014, Vol. 47, No. 6, pp. 285– 287

[3] A. Ivaschenko, A. Kolsanov, A. Nazaryan and A. Kuzmin “3D surgery simulation software development kit”. *Proceedings of ESM 2015*, Leicester, UK, EUROSIS-ETI, 2015, pp. 333 – 240

[4] D.L. Rodgers et al., “The effect of hi-fi simulation on educational outcomes”, *Simulation in Healthcare*, 2009, Vol. 4, pp. 200 – 206

[5] F. Bello and H. Brenton “Current and future simulation and learning technologies”, *Surgical education advances in medical education*, 2011, Volume 2, pp. 123 – 149

[6] R. Owens and J.M. Taekman, “Virtual reality, haptic simulators, and virtual environments”. The comprehensive textbook of healthcare simulation. Springer New York, 2013, pp. 233 – 253

[7] D.A. McClusky III and C.D. Smith, “Design and development of a surgical skills simulation curriculum”, *World Journal of Surgery*, 2008, Volume 32, Issue 2, pp. 171 – 181

[8] C. Karaliotas “When simulation in surgical training meets virtual reality”, *Hellenic Journal of Surgery*, 2011, 83: 6, pp. 303 – 316

[9] M. Zhou, S. Tse, A. Derevianko, D.B. Jones, S.D. Schwaitzberg and C.G.L. Cao, “Effect of haptic feedback in laparoscopic surgery skill acquisition”, *Surgical Endoscopy*, 2012, Vol. 26, Iss 4, pp.1128-1134

[10] M.C. Cavusoglu, T.G. Goktekin, F. Tendick and S.S. Sastry, “GiPSi: an open source/ open architecture software development framework for surgical simulation”. *Proceedings of Medicine Meets Virtual Reality XII (MMVR 2004)*, Newport Beach, CA, 2004, pp.46 – 48

[11] J. Allard, S. Cotin, F. Faure, P. Bensoussan, F. Poyer, et al., “SOFA - an open source framework for medical simulation”. *Proceedings of Medicine Meets Virtual Reality 15 (MMVR 2007)*, Palm Beach, United States. IOP Press, 2007, 125, pp. 13 – 18