(12) **APPLICATION**

(11) **20180688** (13) **A1**

(19) NO

NORWAY (51) Int Cl.

A63F 13/577 (2014.01)

Norwegian Industrial Property Office

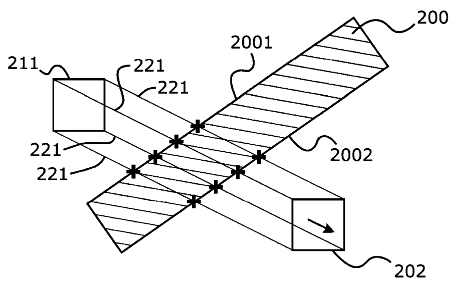
(21)	Application nr	20180688	(86)	Int. application day and application nr	
(22)	Application day	2018.05.15	(85)	Entry into national phase	
(24)	Date from which the industrial right has effect	2018.05.15	(30)	Priority	
(41)	Available to the public	2019.11.18			

(71)	Applicant	Markus Fjellheim, Kjærholen 77, 4316 SANDNES, Norge
(72)	Inventor	Markus Fjellheim, Kjærholen 77, 4316 SANDNES, Norge
(74)	Agent or Attorney	HÅMSØ PATENTBYRÅ AS, Postboks 171, 4301 SANDNES, Norge

(54) Title **METHOD OF RENDERING A FRAME IN A DISCRETE PHYSICS ENGINE FOR SIMULATING RIGID BODIES IN A SPACE**

(57) Abstract

Disclosed is a computer-implemented method of rendering a frame in a discrete physics engine for simulating rigid bodies in a space. The method comprises the steps of: for each object being simulated, processing a new position based on at least the position and the velocity of the object; updating the position of a copy of an object to match the position of the object, if no collision has been detected between the object and another object while rendering the previous frame; and detecting a collision by processing if there is an intersection between a first object at its new position and a line segment between a copy of a second object and the second object at its new position.



METHOD OF RENDERING A FRAME IN A DISCRETE PHYSICS ENGINE FOR SIMULATING RIGID BODIES IN A SPACE

The present invention relates to a method of rendering a frame in a discrete physics engine for simulating rigid bodies in a space.

A physics engine is a computer program that provides an approximate simulation of certain physical systems by simulating Newtonian (or other) physics models so that simulated objects behave as obeying the laws of physics. Nowadays, physics engines are used for multiple purposes, such as analysing rigid body dynamics, and in various domains, such as computer graphics, video games or film production. Physics engines typically include methods for detecting collisions.

Collision detection in a physics engine relates to the computational problem of detecting the overlapping of two or more objects. Depending on the manner in which collisions are detected, physics engines may be classified in one of two ways: where the collision is detected *a priori*, ie. before the collision occurs, or *a posteriori*, ie. after a collision occurs. These two classes of physics engines are also known as continuous and discrete, respectively.

In a discrete physics engine, the simulation is advanced by a time step, then a check is made as to whether any objects are overlapping (eg. using the separating axis theorem) or are so close to each other that they are deemed to be intersecting. This method is said to be *a posteriori* (ie. based on reasoning from known facts or past events) because it typically misses the instant of collision and only detects the collision after it has happened in simulation time.

Some of the advantages of a discrete physics engine are: typically, it requires less processing resources than a continuous physics engine; it does not take into account a lot of physical variables about the simulated environment in which the rigid bodies are moving; and it does not need to take into account friction, elastic collisions, or non-elastic collisions and deformable bodies.

It can be challenging to detect a collision in a discrete physics engine. Each frame is treated separately, and the position of an object between frames is not calculated. A low frame rate and a small fast-moving object can cause a situation where the object does not move smoothly through space but instead seems to teleport from one point in space to the next as each frame is calculated. Thus, a projectile being simulated to move at a sufficiently high speed will miss a target object, if the latter is small enough to fit in the gap between the calculated frames of the projectile. This is usually known as the tunnelling problem. Also, it is usually said that a projectile tunnels through a target object when the first misses a collision with the latter due to the tunnelling problem.

Figure 1 shows a schematic view of several frames generated by a discrete physics engine while simulating a two-dimensional environment. Two squares move in the horizontal trajectories 110 and 120 from left to right (see time axis 102 at the top of the Figure). Both trajectories intersect the rectangle 101 that is standing upright at a fixed position on the ground 100.

The squares of the two trajectories 110 and 120 were simulated moving at different velocities: the square of the trajectory 110 had a lower velocity than the square of the trajectory 120.

In the trajectory 110, the frames are processed one after another as follows: first, square 111 is processed, then square 112, and afterwards square 113; and then, when the subsequent square 114 is processed, the physics engine will detect a collision with rectangle 101 because both objects intersect.

However, in the trajectory 120, the square is moving too fast and the collision is not detected. The processing of the first two frames for the trajectory 120 results in squares 121 and 122. However, when the subsequent frame is processed, the square 123 will be positioned on the right-hand side of the rectangle 101 without both objects intersecting each other. In known solutions, the trajectory 120 will wrongfully result in that a collision is not detected.

A known solution is to define global constraints on the simulated environment, such as limiting the velocity at which objects can move so that the objects do not move fast enough to completely tunnel through other objects. Another known constraint is to make the main objects large enough so that the fastest moving objects could never completely tunnel though them. This solution puts restrictions on the environment being simulated, and that is not a viable solution most of the times because the scenarios that can be simulated are greatly reduced.

Another known solution is to increase the frame rate. This solution tackles the tunnelling problem by reducing the gap between the positions of an object in two frames in sequence. However, this solution can still be insufficient if an object moves sufficiently fast, and it can be wasteful if in most of the simulated time the objects are moving slowly, which does not require the additional simulation granularity for the collisions to be detected.

The present invention will now be disclosed.

According to an aspect of the present invention, there is provided a computer-implemented method of rendering a frame in a discrete physics engine for simulating rigid bodies in a space, the method comprising the steps of:

- providing at least one data structure for representing an object, each data structure comprising a position and a velocity of the object in the space;
- providing, for each data structure representing an object, at least one second data structure for representing a copy of the object, each second data structure comprising a position of the copy of the object in the space;
- for each object, processing a new position based on at least the position and the velocity of the object;

- updating the position of a copy of an object to match the position of the object, if no collision has been detected between the object and another object while rendering the previous frame;
- detecting a collision by processing if there is an intersection between a first object at its new position and a line segment between a copy of a second object and the second object at its new position.

Said line segment may be a line segment between a corner of the copy of the second object and the corresponding corner of the second object. Also, in the step of detecting a collision, the collision may be detected if there is an intersection between an edge of the first object and the line segment. The collision may also be detected if there is an intersection between a face of the first object and the line segment.

The method may comprise the steps of:

- generating a new position for the copy of the second object, the new position of the copy being nearer to the new position of the second object;
- updating the position of the copy of the second object to match the generated new position of the copy, if at that new position the copy does not collide with the copy of the first object.

In the step of generating a new position for the copy of the second object, the new position may be a random position between the position of the copy and the new position of the second object.

Moreover, each data structure representing an object may comprise a variable for storing whether a collision has been detected between the object and another object when rendering the previous frame. Also, each second data structure for representing a copy of the object may be comprised by the data structure representing the object. Moreover, the at least one second data structure may be provided so that, for each unordered pair of objects, there are two second data structures, each paired object having a copy of itself being represented by one of the two second data structures. Furthermore, the at least one data structure for representing an object may comprise at least one acceleration of a point of the object.

According to other aspects of the present invention, there is provided a data processing device comprising means for carrying out the steps of the method. There is also provided a computer program comprising instructions which, when the program is executed by a computer, cause the computer to carry out the steps of the method. Moreover, there is provided a computer-readable storage medium comprising instructions which, when executed by a computer, cause the computer to carry out the steps of the method.

Embodiments of the invention will now be described, by way of example only, with reference to the accompanying drawings, in which:

Figure 1 is a schematic view of two objects moving at different velocities, in which the tunnelling problem is illustrated with a faster object tunnelling through an obstacle;

Figure 2 is a schematic view of a frame rendered by a discrete physics engine while simulating a two-dimensional environment in accordance with a method embodiment, the frame showing two objects that are set to collide with each other;

Figures 3, 6, and 9 are schematic views of intermediate states of the simulated environment, in which each object has a new position that has been processed based on its position and velocity;

Figures 4, 7, and 10 are schematic views of further intermediate states of the simulated environment, showing how a collision between the two objects is checked;

Figures 5, 8, and 11 are schematic views of a first, a second, and a third frame, respectively, rendered in sequence after the frame show in Figure 2;

Figure 12 is a schematic view of two objects colliding during a simulation in a discrete physics engine, in which one of the objects is a concave polygon;

Figure 13 is a schematic view of how the simulation in Figure 12 would develop without optimizing the position of the copy of the moving object;

Figure 14 is a schematic view of how the simulation in Figure 12 would develop with optimization of the position of the copy of the moving object.

In one method embodiment, a discrete physics engine simulates two objects colliding in a two-dimensional space. One of the objects is standing still and the other object is moving and colliding with the first.

Figures 2 to 11 show four frames rendered in sequence by the method embodiment. Figure 2 illustrates a frame showing the initial positions and velocities of both objects, and the Figures 3 to 11 are organised in the following sets: Figures 3 to 5 show two intermediate states of the simulated environment and a resulting first frame; Figures 6 to 8 show another two intermediate states of the simulated environment and a resulting second frame; and Figures 9 to 11 show two more intermediate states of the simulated environment and a resulting third frame.

The method embodiment may be carried out by configuring a discrete physics engine and executing the latter in a computational device. The discrete physics engine is configured with two data structures for representing each object, and each data structure includes a position and a velocity of the respective object in the two-dimensional space. Also, for each data structure representing an object, a second data structure is configured in the discrete physics engine representing a copy of the object, and it includes a position of the copy of the object in the space. The data structures may be combined so that the object data and the copy data are stored together in the same data structure. Moreover, each data structure may include other data for characterising the object and/or copy, such as the acceleration of the object, the shape, and the colour or other parameters for configuring the graphical representation of the object/copy.

Each data structure representing an object includes a position and a velocity of the respective object being simulated. That is, each data structure includes the necessary data means for holding data that allows processing a position for the object, such as variables for storing the coordinates values in each dimension and the orientation/rotation values of the object, and a velocity of the object, such as variables for storing the linear or angular velocity components of the object.

In Figure 2, the two objects are shown at their initial positions 201 and 200. The stationary object at position 200 has a null velocity and it will be an obstacle to the moving object at position 201. The latter is shown with an arrow drawn in its interior, representing the direction of its velocity.

<u>Rendering the first frame</u>

Based the positions and velocities of the objects shown in Figure 2, a new position is processed for each object. The object at position 200 has no velocity, and it obtains a new position with the same coordinates as position 200. The object at position 201 does not have a null velocity, and the new position will be different from the position 201.

In Figure 3, the new positions of the objects are shown. The position 202 is the new position of the moving object that is at position 201 in Figure 2 (see curved arrow in Figure 3 showing the position update), and the new position of the object at position 200 in Figure 2 is the same.

There are known methods for processing the new position 202. A simple approach is to calculate which position will the moving object have if it travels with constant velocity, starting from the position 201, during the time step defined by the physics engine. Other kinematic models may include different aspects in this calculation, such as medium (eg. air, water) friction or a gravity force.

In a discrete physics engine, no consideration is taken as to whether a collision will occur between objects before processing new positions. It is only after the new positions have been processed that it is checked if there are any objects colliding when these are at the new positions. That is, the detection of collisions is performed after the simulation has been advanced by a time step.

The new positions 200 and 202 processed for the two objects provide the conditions for a tunnelling problem to happen. The two objects do not overlap at these new positions 200 and 202, however, when considering the motions of the two objects from the positions 200 and 201 to the positions 200 and 202, respectively, the moving object performs an imaginary passage through the stationary object. Thus, in order to avoid the tunnelling problem in this case, a collision must be detected.

Figure 4 shows how the method embodiment detects the collision between the two objects.

The method embodiment starts by updating the position of the copy of each object if no collision has been detected involving the object while rendering the previous frame, ie. the frame shown in Figure 2. No collision has been detected for the moving object while rendering the previous frame, and, thus, the copy of the moving object is set with the position 211, which is the position 201 that the object had in the previous frame.

Then, four line segments 221 are processed between the position 211 of the copy and the new position 202. Each line segment 221 starts from a corner of the copy of the moving object at the position 211 and ends at the corresponding corner of the moving object at the new position 202.

After this, the line segments 221 are processed to verify if they intersect with any edge of the stationary object at position 200. In this case, it is verified that the line segments 221 intersect with the edges 2001 and 2002 (see crosses drawn in Figure 4 at the intersection points), and, therefore, a collision between the moving object at position 202 and the stationary object at position 200 is detected.

In order to simplify the present description, only the copy of the moving object is mentioned, as it is sufficient for detecting the collision and avoiding the tunnelling problem.

This approach is advantageous as it avoids the tunnelling problem and it allows checking if there are any collisions in an efficient manner, requiring only the verification of intersections involving line segments.

Once the detection of the collision has been processed, the velocity of the moving object is changed, and the resulting frame is rendered. In Figure 5, the rendered first frame is shown, including the moving object at is new position 202 with its new velocity (see the arrow inside it, representing the velocity of the object). This velocity was processed using the velocity of the object at the position 201 and the inversion of the angle that the velocity vector makes on the edge 2001 of the object at position 200 (see Figure 4).

In practice, the first frame (Figure 5) is not rendered with the copy at position 202 being shown. Also, if the frames rendered by the method embodiment had been displayed in a computer screen, the frame in Figure 5 would have been shown subsequently to the frame in Figure 2, leaving the intermediate states, illustrated by Figures 3 and 4, hidden.

<u>Rendering the second frame</u>

The steps of the method embodiment that were used for rendering the first frame shown in Figure 5, will now be repeated to render the second frame. However, now the method embodiment also has the data indicating the position 211 of the copy of the moving object and that there was a collision detected when rendering the frame in Figure 5.

First, the simulation is advanced one time step, and new positions are processed for the objects based on the positions and velocities provided on Figure 5. In Figure 6, the object at position 200 is shown standing in the same position as before since it has a null velocity. The new position 203 has been processed for the moving object (see the curved arrow passing over the object 200 represents the position update).

As before, the new positions are now processed to analyse if there are any collisions between the two objects. This step starts by updating the position of the copies of the objects for which no collision was detected when rendering the previous frame (Figure 5). Thus, the copy at position 211 keeps the same position, as there was a collision detected when rendering the previous frame (see Figure 4). Then, the line segments 222 between the copy and the moving object at position 203 are processed. In Figure 7, the line segments 222 are shown, and these start from a corner in the copy at position 211 and end on the corresponding corner of the moving object at position 203. The line segments 222 are then checked for intersections with any edge of the object at position 200, and no intersection is detected. Thus, it is concluded that no collision has occurred.

Since there was no collision detected, the velocity of the moving object at position 203 is not changed, and the second frame is rendered as shown in Figure 8.

When looking at the last three frames in sequence (see Figures 2, 5, and 8, respectively), it is possible to observe the simulated trajectory of the moving object. In the middle frame, the moving object is shown at position 202, appearing to be doing the same as it would do in case the tunnelling problem had happened. However, the method embodiment solves the tunnelling problem by having the velocity of the object at position 202 being affected by the collision detected with the object 200. Thus, this will have a correcting effect when rendering the next frames, and the last frame in Figure 8 shows the object at position 203 on the correct side of the object 200.

Rendering the third frame

At the moment of rendering the last frame in Figure 8, the copy at position 211 continues to have a different position than the object at position 203. However, since no collision has been detected when rendering the last frame (see Figure 7), the position 211 of the copy will be updated when rendering the next frame.

In Figure 9, the moving object is shown at its new position 204, which was processed based on the position and velocity of moving object at position 203. Also, as before, the object 200 remains at the same position due to its null velocity.

The method embodiment will now, as before, check if there is any collision between the two objects (see Figure 10). First, the copy at position 211 will be updated to match the position 203, since there was no collision detected when rendering the previous frame, shown in Figure 8. The copy at the matched position 212 is shown in Figure 10. Then, the line segments 223 are generated, as before, and it is verified if the line segments 223 intersect with any of the edges of the object at position 200. As no intersection is detected, it is concluded that no collision is detected between the moving object and the stationary object.

The method embodiment thus moves on to render the third frame, as shown in Figure 11.

When comparing the three intermediate states for detecting a collision (ie. Figures 4, 7, and 10, respectively), it can be observed that the copy of the moving object is kept at the position 211 until a collision involving the moving object is no longer detected. Only then is the copy set with the new position 212.

This results in that a collision is detected as long as the moving object stays on the wrong side of the stationary object at position 200, and thus the tunnelling problem is avoided.

For the purposes of simplifying the present description, the situation shown in the Figures 2 to 11 is simple and has only two objects. However, in practice many more objects may be simulated and presented in frames rendered by a method embodiment. Also, the objects shown are convex polygons, however in practice other polygons may be simulated, such as concave polygons.

When simulating more than two objects, the updating of the positions of each copy can vary. One approach is to update the position of the copy only when the copied object is no longer detected to be colliding with any other copies being simulated. Another approach is to provide a copy of an object for each collision that an object may perform with the other objects being simulated, having the position of each copy being updated if the collision corresponding to that copy is not detected. In order to carry out this approach, the at least one second data structure may be provided so that, for each unordered pair of objects, there are two second data structures, each paired object having a copy of itself being represented by one of the two second data structures.

Correction of the position of a copy of an object

When simulating objects more complex than the ones shown in Figures 2 to 11, it may happen that the position of a copy of an object can be optimised. Figure 12 shows a stationary object, at position 300, and a moving object, at position 301, however the stationary object at position 300 presents a concave shape. The moving object at position 301 has a velocity towards the interior area of the stationary object at position 300 (see arrow inside the moving object at position 301). After a new position 302 is processed for the moving object, the method embodiment will detect a collision as shown in Figure 12: the line segment 321 between a corner of a copy of the moving object at position 301 and the corresponding corner of the moving object at position 302 intersects with an edge of the stationary object at position 300 (see cross drawn at the intersection point). Thus, a collision is detected in this case. Also, a change in the velocity of the object at position 302 is processed (compare arrows inside the moving object at positions 301 and 302).

Without optimisation of the position of the copy of the moving object at the position 301, the method embodiment will simulate the situation shown in Figure 13. The copy of the moving object will remain in position 301, because a collision was detected when the object was at position 302 (see Figure 12). Also, the object will move to the new position 303 due to the new velocity. However, the detection of a collision will find the intersections between the line segments 322 and the edges of the stationary object 300 (see crosses at the intersection points). This detected collision influences the update of the velocity of the moving object, as if the moving object had moved from position 301 to position 303, without having passed through position 302.

With optimisation of the position of the copy of the moving object at the position 301, the method embodiment will simulate the situation shown in Figure 14, instead of Figure 13. The position 312 of the copy of the moving object was generated after the collision in Figure 12 had been detected. The generated position 312 is nearer to the position 303 of the moving object. Since the generated position 312 did not collide with the stationary object at position 300, the position of the copy of the moving object was updated.

One method of generating the position 312 such that the copy of the moving object becomes nearer to the position 303 of the moving object is by choosing a random position between the position 301 of the copy and the position 302 of the second object.

Embodiments of the invention may have some or all of the following advantages:

- Avoids the tunnelling problem while rendering a frame in a discrete physics engine

- Efficient collision detection based on intersections between line segments and polygon edges
- Detects collisions involving concave polygons

Generally, the terms used in this description and claims are interpreted according to their ordinary meaning the technical field, unless explicitly defined otherwise. Notwithstanding, the terms “comprises” and “comprising” and variations thereof mean that the specified features, steps or integers are included. These terms are not interpreted to exclude the presence of other features, steps or integers. Furthermore, the indefinite article “a” or “an” is interpreted openly as introducing at least one instance of an entity, unless explicitly stated otherwise. An entity introduced by an indefinite article is not excluded from being interpreted as a plurality of the entity.

The features disclosed in the foregoing description, or in the following claims, or in the accompanying drawings, expressed in their specific forms or in terms of a means for performing the disclosed function, or a method or process for obtaining the disclosed results, as appropriate, may, separately, or in any combination of such features, be utilised for realising the invention in diverse forms thereof.

While the invention has been described in conjunction with the embodiments described above, many equivalent modifications and variations will be apparent to those skilled in the art when given this disclosure. Accordingly, the embodiments of the invention set forth above are considered to be illustrative and not limiting. Various changes to the described embodiments may be made without departing from the spirit and scope of the invention.

CLAIMS

1. A computer-implemented method of rendering a frame in a discrete physics engine for simulating rigid bodies in a space, the method comprising the steps of:

- providing at least one data structure for representing an object, each data structure comprising a position and a velocity of the object in the space;
- providing, for each data structure representing an object, at least one second data structure for representing a copy of the object, each second data structure comprising a position of the copy of the object in the space;
- for each object, processing a new position (202, 203, 204, 302, 303) based on at least the position (201, 202, 203, 301) and the velocity of the object;
- updating the position (211, 212, 301) of a copy of an object to match the position of the object, if no collision has been detected between the object and another object while rendering the previous frame;
- detecting a collision by processing if there is an intersection between a first object at its new position and a line segment (221, 222, 223, 321, 322, 323) between a copy of a second object and the second object at its new position.

2. A method according to claim 1, wherein said line segment is a line segment between a corner of the copy of the second object and the corresponding corner of the second object.

3. A method according to any of the previous claims, wherein, in the step of detecting a collision, the collision is detected if there is an intersection between an edge of the first object and the line segment.

4. A method according to any of the claims 1 to 2, wherein, in the step of detecting a collision, the collision is detected if there is an intersection between a face of the first object and the line segment.

5. A method according to any of the claims 1 to 4, the method comprising the steps of:

- generating a new position for the copy of the second object, the new position of the copy being nearer to the new position of the second object;
- updating the position of the copy of the second object to match the generated new position of the copy, if at that new position the copy does not collide with the copy of the first object.

6. A method according to claim 5, wherein, in the step of generating a new position for the copy of the second object, the new position is a random position between the position of the copy and the new position of the second object.

7. A method according to any of the previous claims, wherein each data structure representing an object comprises a variable for storing whether a collision has been detected between the object and another object when rendering the previous frame.

8. A method according to any of the previous claims, wherein each second data structure for representing a copy of the object is comprised by the data structure representing the object.

9. A method according to any of the claims 1 to 7, wherein the at least one second data structure is provided so that, for each unordered pair of objects, there are two second data structures, each paired object having a copy of itself being represented by one of the two second data structures.

10. A method according to any of the previous claims, wherein the at least one data structure for representing an object comprises at least one acceleration of a point of the object.

11. A data processing device comprising means for carrying out the steps of the method of any of claims 1 to 10.

12. A computer program comprising instructions which, when the program is executed by a computer, cause the computer to carry out the steps of the method of any of the claims 1 to 10.

13. A computer-readable storage medium comprising instructions which, when executed by a computer, cause the computer to carry out the steps of the method of any of the claims 1 to 10.

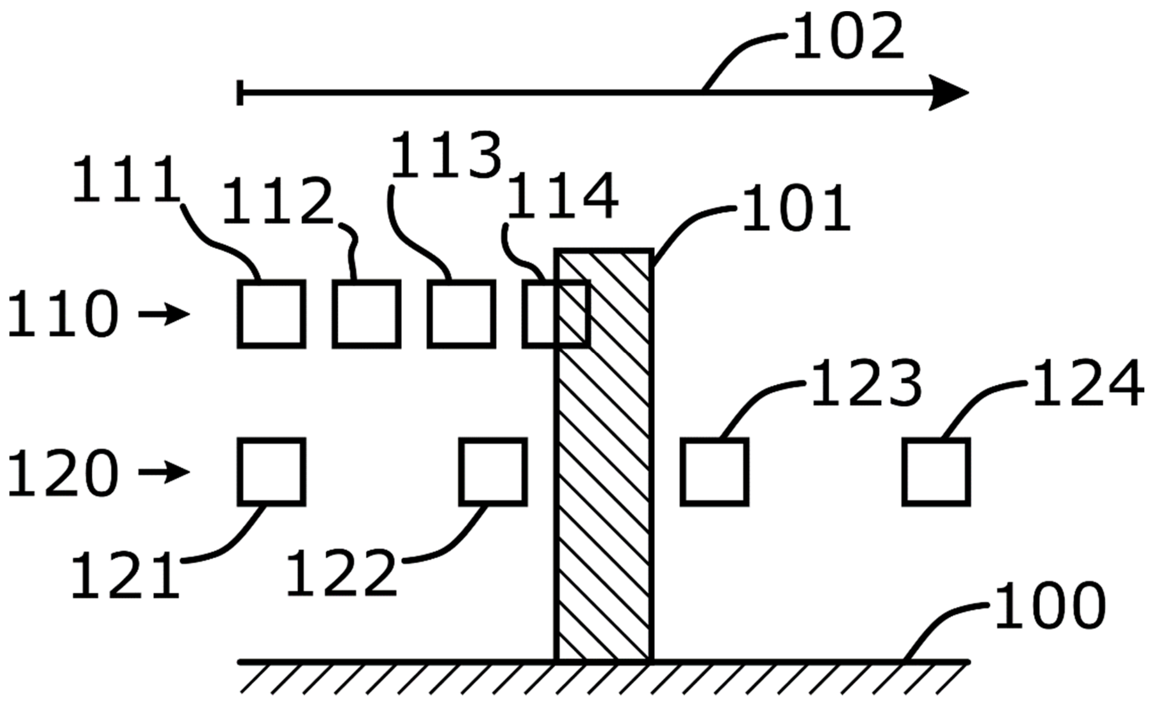


FIG. 1

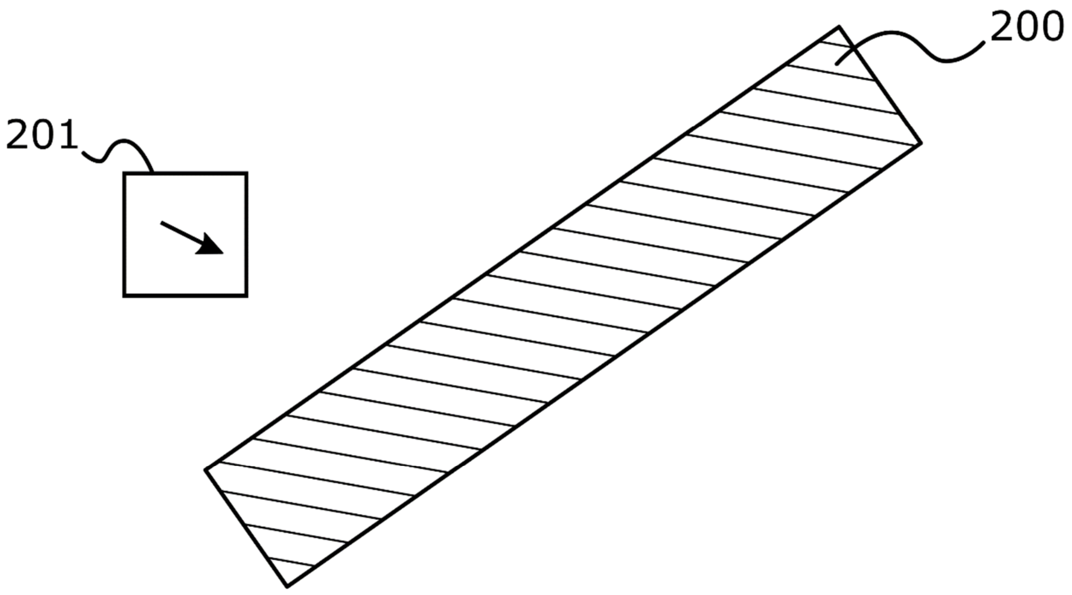


FIG. 2

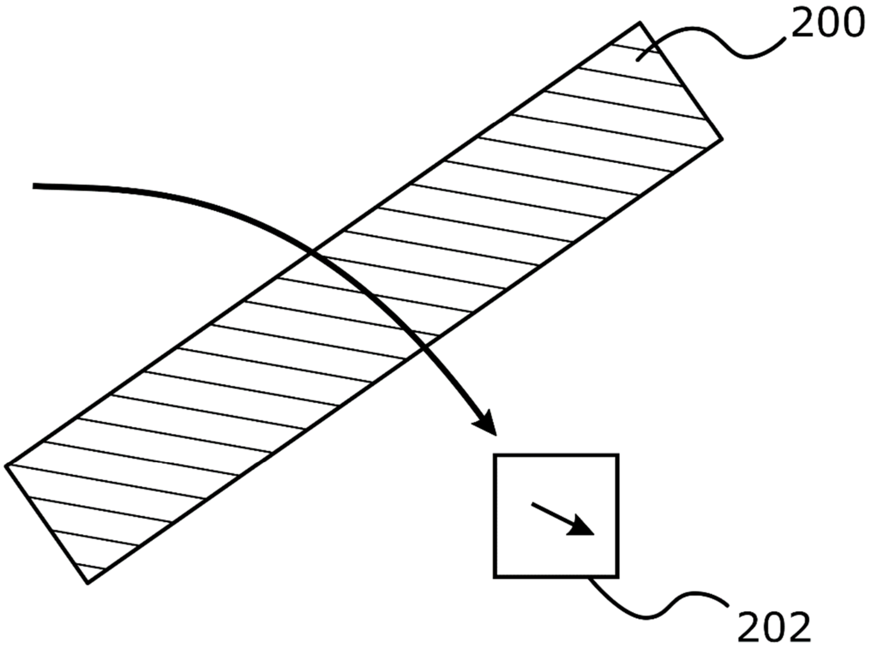


FIG. 3

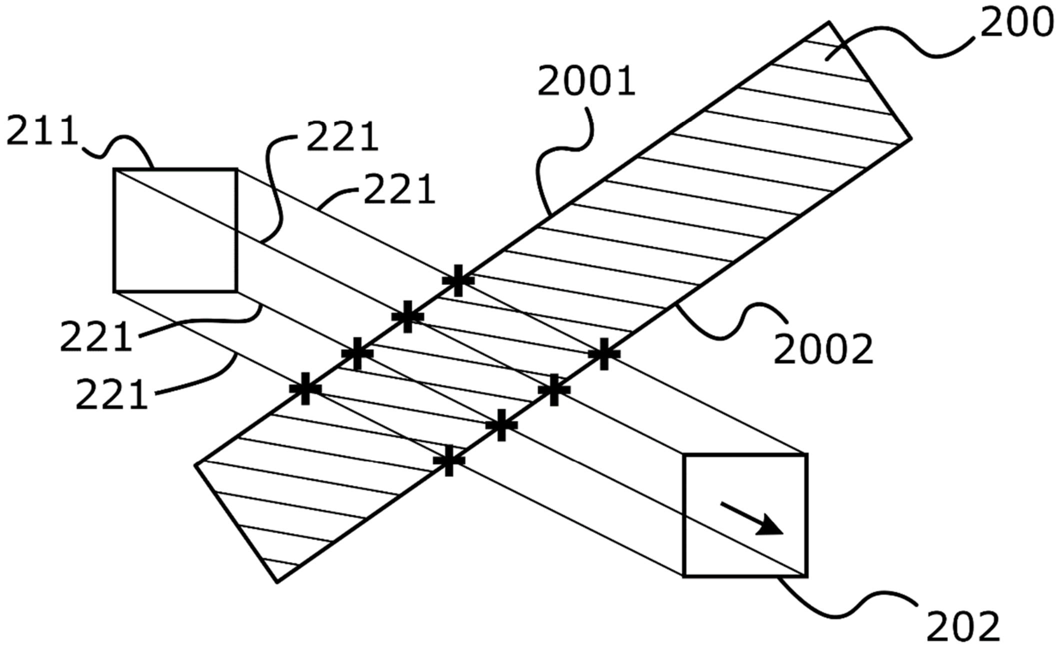


FIG. 4

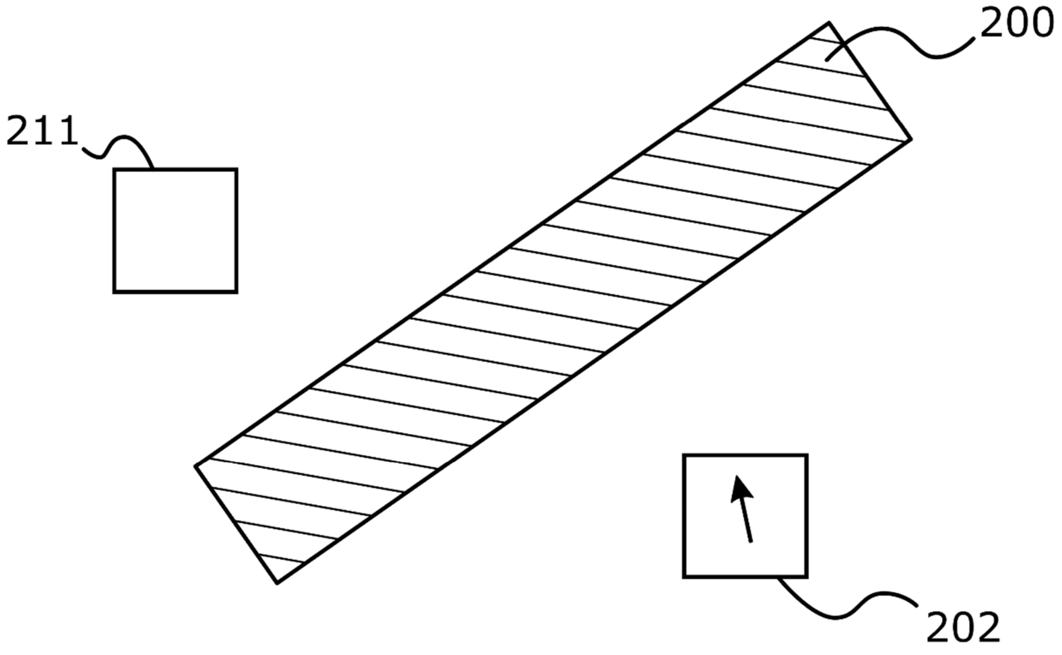


FIG. 5

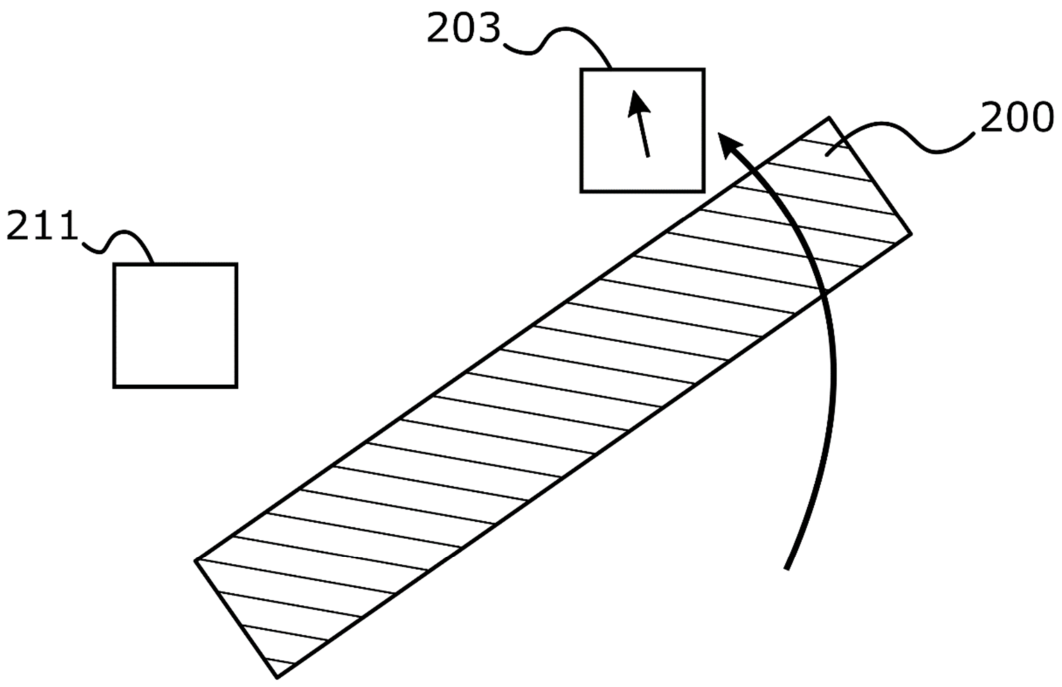


FIG. 6

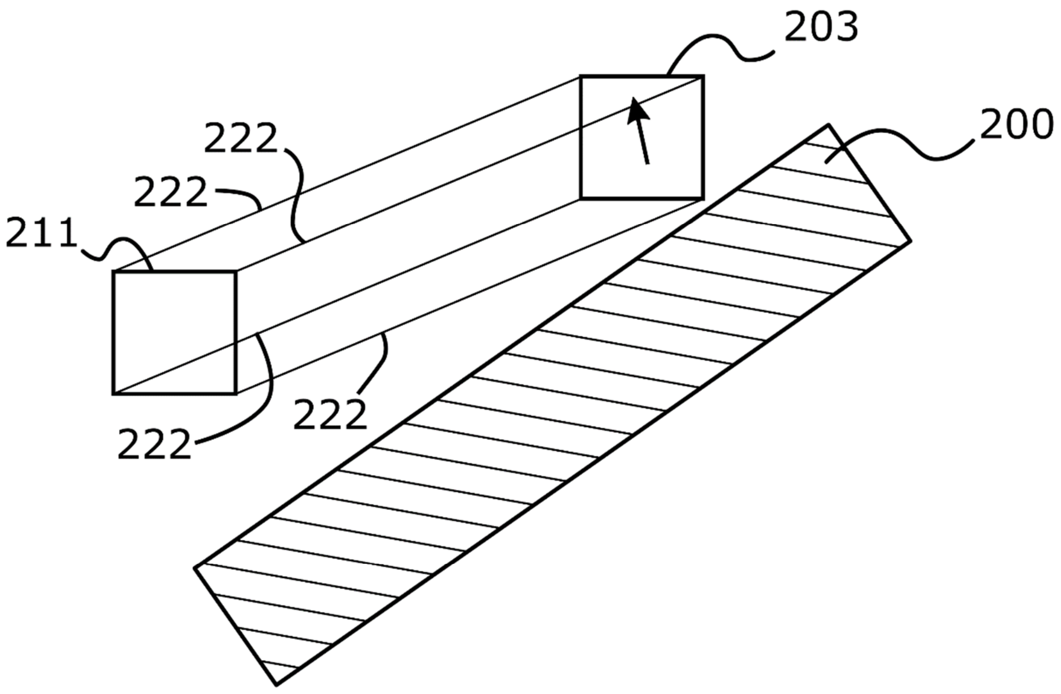


FIG. 7

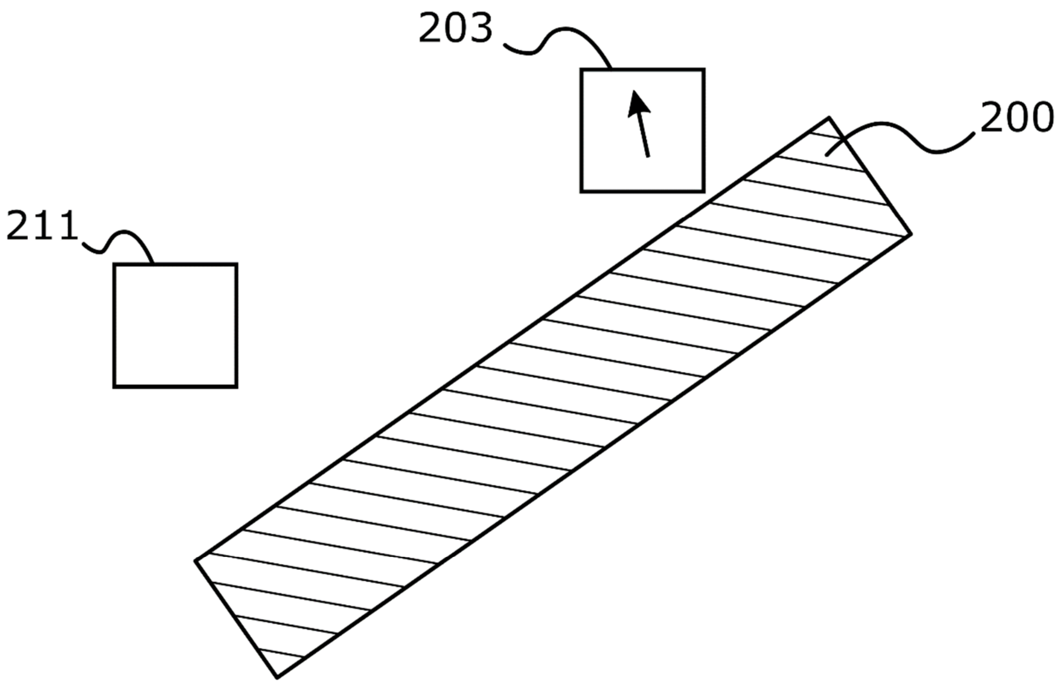


FIG. 8

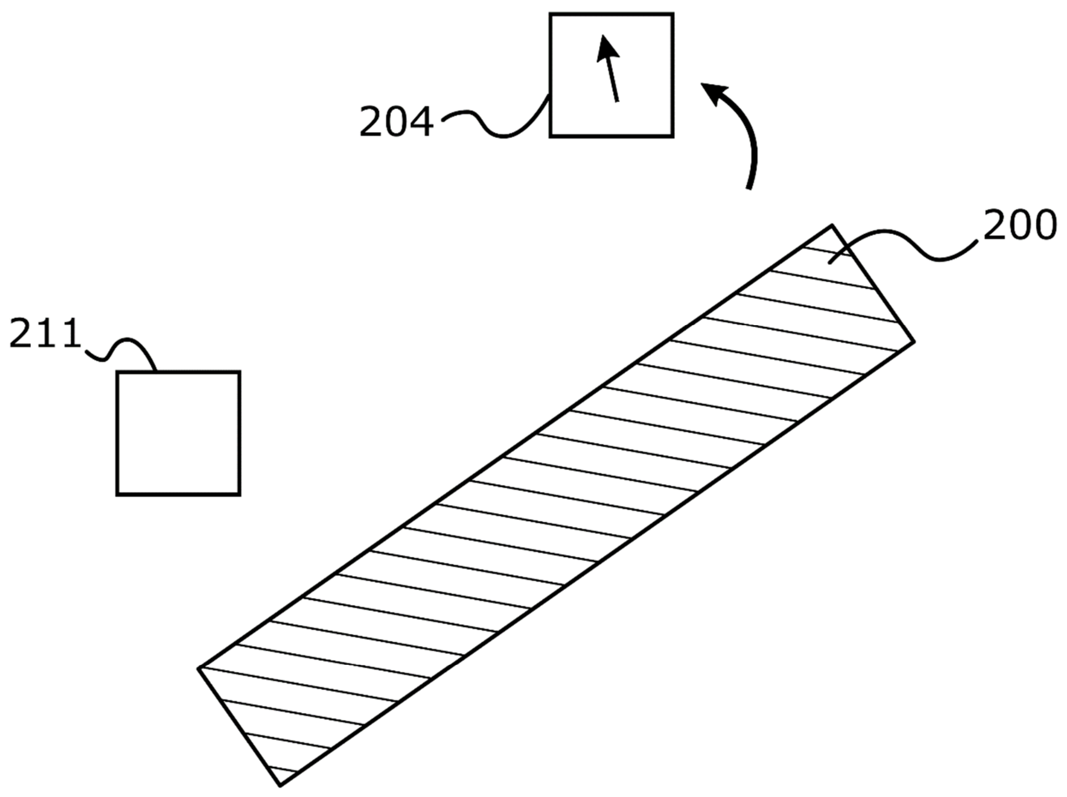


FIG. 9

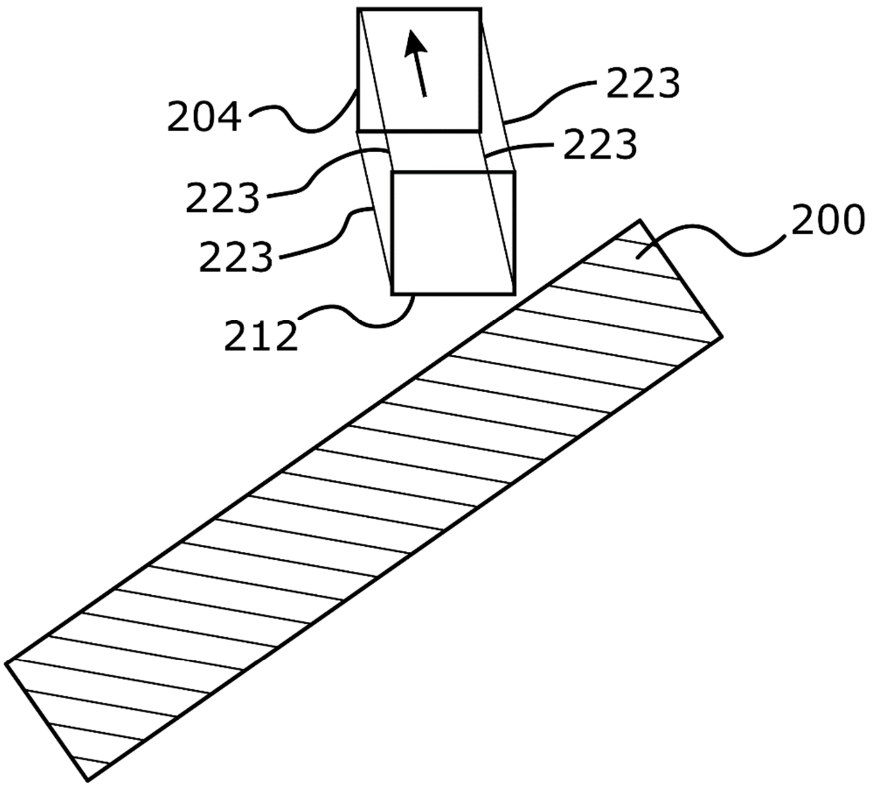


FIG. 10

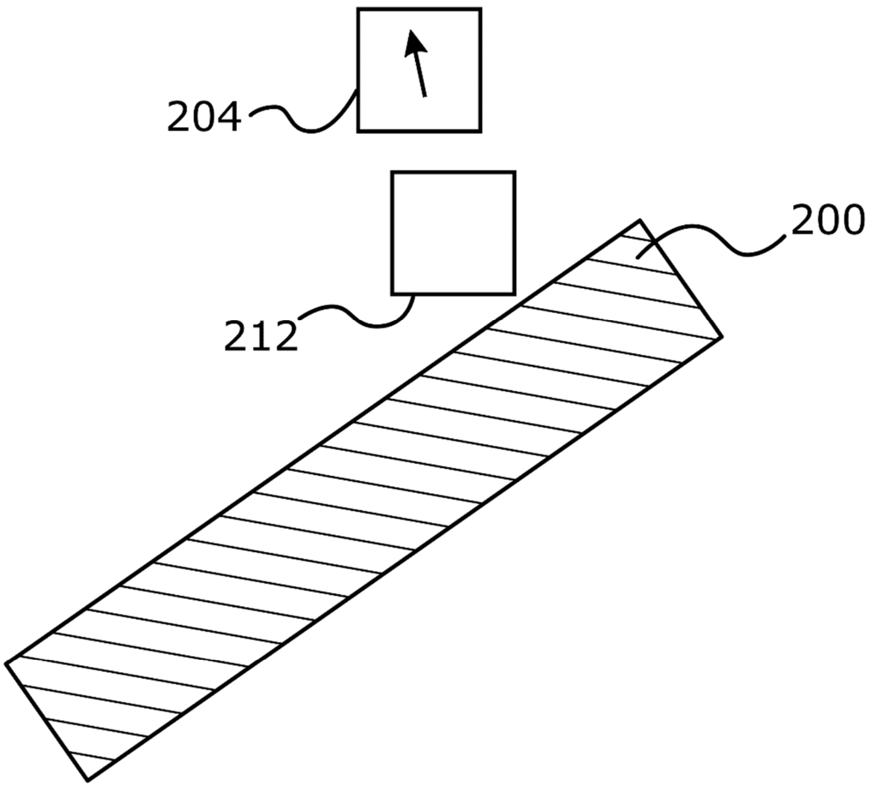


FIG. 11

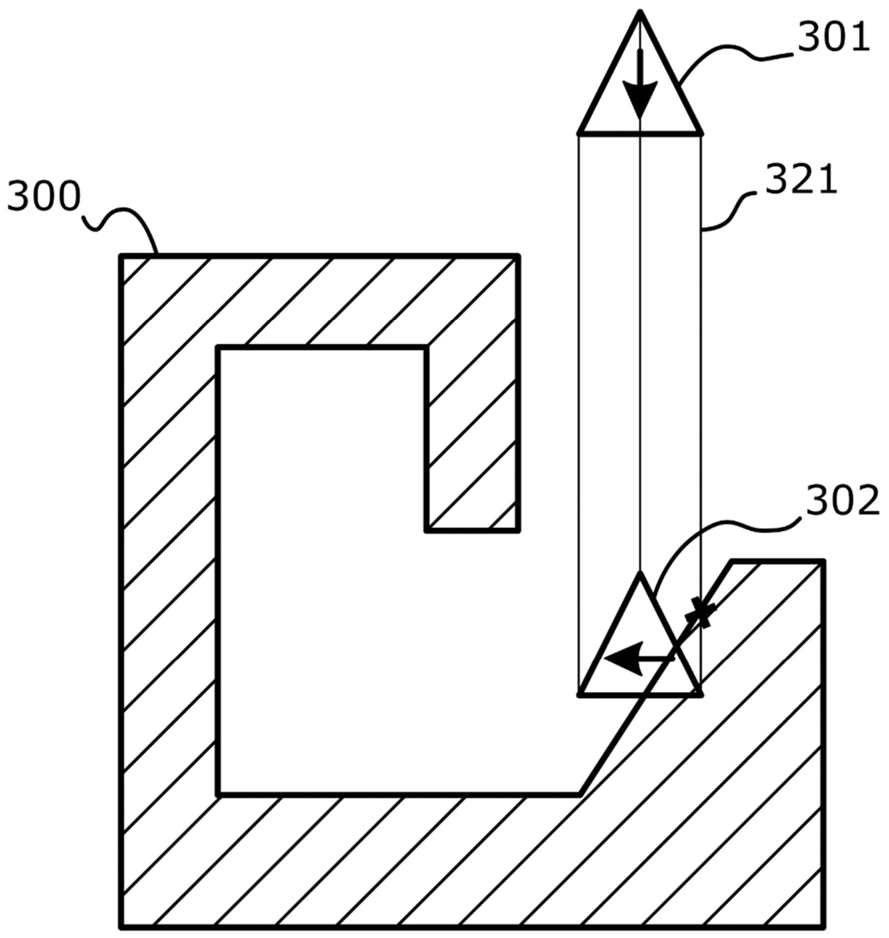


Fig. 12

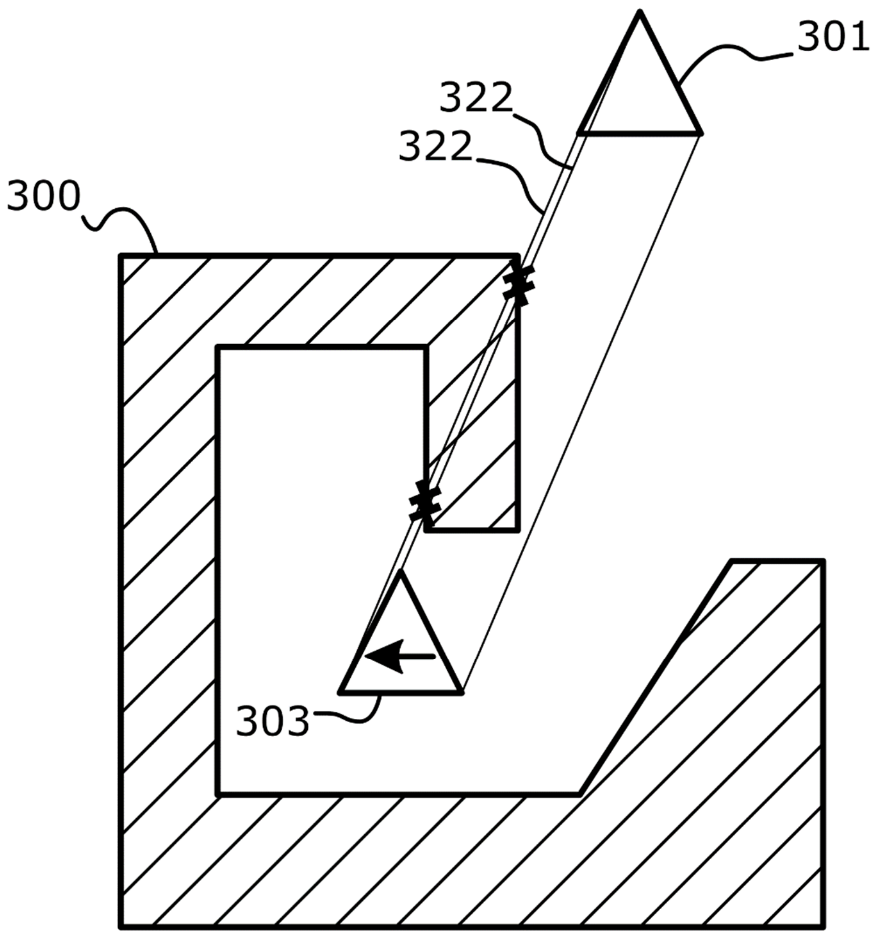


Fig. 13

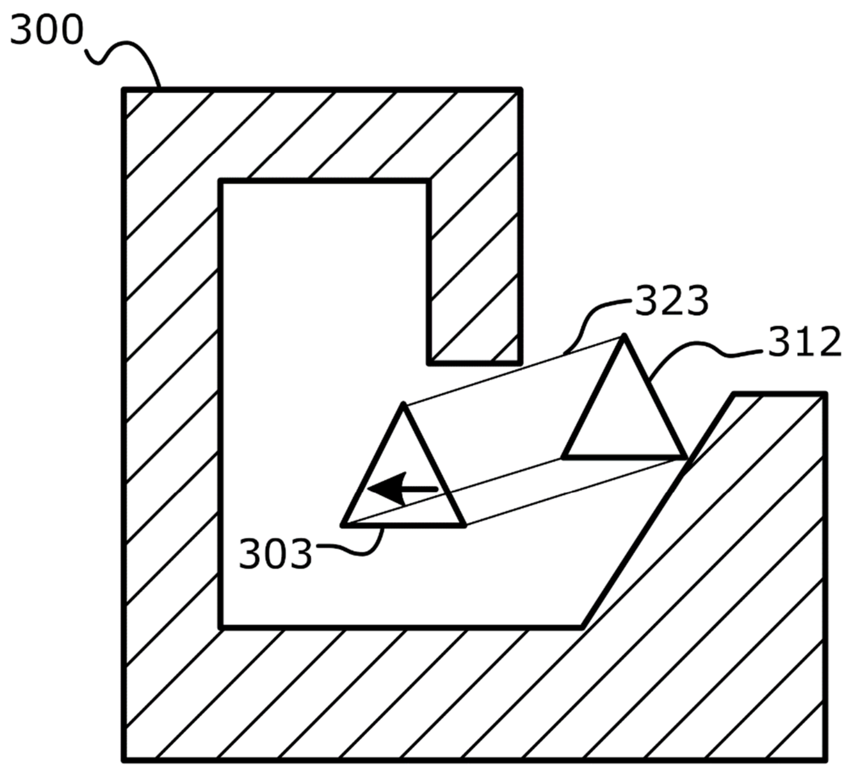


Fig. 14