Placement Estimation Overhaul

Maximilian David
s1madavi 1349951
maximilian.david@uni–bayreuth.de

November 23, 2022

The placement estimation in the software version that was used during the pilot studies was performed the following way: A ray was cast down from the main anchor of the brick's model to the build plate. This resulted in an intersection point of the ray and the XZ - plane that is the base plate. The X and Z coordinate values of the grid's origin were then subtracted from the point's X and Z coordinate to gain X and Z coordinates relative to the origin of the grid system. These values were then divided by the size of each grid square to calculate the intersection point's position inside the grid.
To determine the grid square inside which the intersection point is located in integer format, for later indexing, the floor operator was used. Then the brick model's main anchor was set to the coordinates of the calculated grid square. Effectively snapping the brick's main anchor, which is simultaneously the model's origin, to the origin of the calculated grid square. While this resulted in an always correct calculation of the grid square which contains the intersection point, it also resulted in a snapping behavior that did not match user expectations.

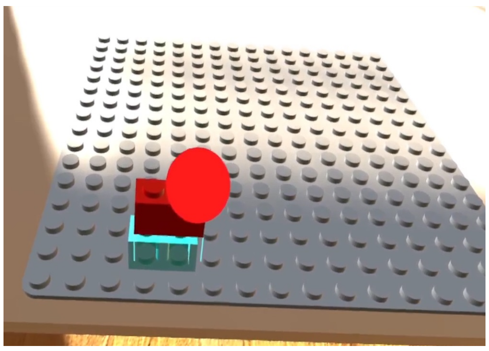

Figure 1: Estimation with old system

To improve the ease of use whilst working with the software's code, a new class "SnapPoint" was created to represent a snap point. An object of this class provides the exact position of the snap point in world coordinates, in grid coordinates, as well as the number of the corresponding grid. This provides far more information as the mere world location that was used before the introduction of the "SnapPoint" class and makes many redundant calculations obsolete.

To improve the naturalness and align the system's functionality with the user's expectations, it was determined that the estimation should not return the grid field's coordinates that contain the intersection point, but rather the closest grid anchor point. This is realistically the most likely position the user intended.
This worked exponentially better than the calculation method used prior, while the brick remained in its default rotation. However, if the brick's rotation was changed, the now changed estimation function would still try to line up the brick's main anchor to the closest grid anchor. This resulted in drastically unnatural snapping in every rotation, other than default, due to the model's origin being located at one of the brick's corners and not in its center.

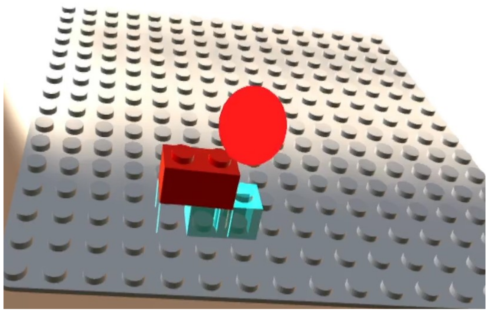

Figure 2: New estimation with brick in non default rotation

To mitigate this problem, a function was created that, depending on the brick's rotation, returns the brick's anchor which should be used for placement estimation for the current rotation. Now the correct anchor would be aligned to the grid's anchor point. But since these other anchors are not the brick model's origin, the resulting coordinates need to be translated first, to accurately snap the brick into the grid correctly. This is simply done by shifting the estimated origin position by the difference between the main anchor's location and anchor's location used for estimation.

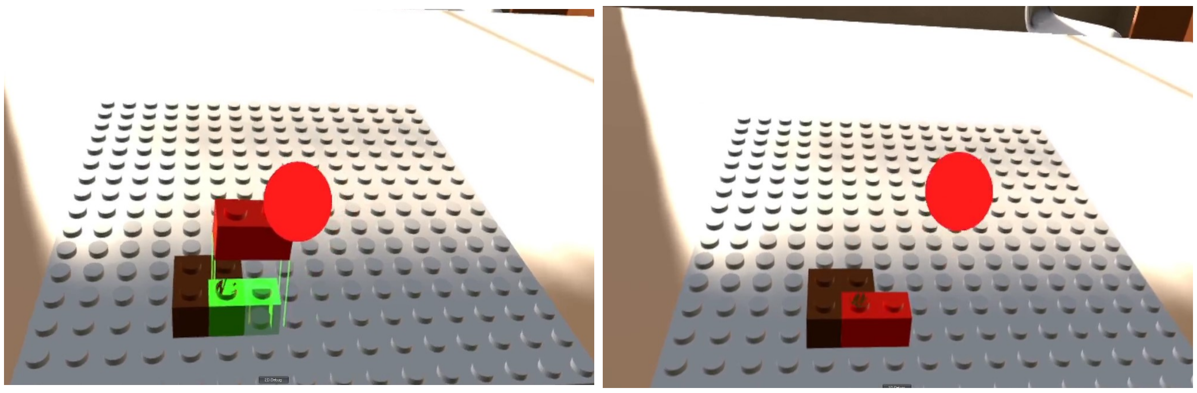

Figure 3: Erroneous placement estimation inside other bricks

Now, the placement estimation felt correct, but by simply shifting the brick's position within the grid, the checks for available space were now erroneous. This resulted in estimations inside other bricks, which rendered the entire system unusable in it's state. Contrary to the impact of this problem, the corresponding fix was very low scale. The checks for available grid space simply needed to be executed after the shift had taken place and not beforehand.

This achieved a form of placement estimation that feels natural and correct, which indicates a match between the user's expectations and the software's functionality. However, developing the placement estimation in the described way led to a new immersion breaking bug. If the user tried to place a brick next to an already placed brick, and the intersection point would lie on top of the already placed brick, the newly placed brick would float above its intended position. This phenomenon was caused due to the fact that the placement estimation tried to find an unoccupied grid position within the same grid as the intersection point. This, in turn, caused the new brick to be placed in mid air, since the estimated grid had the appropriate space for it.

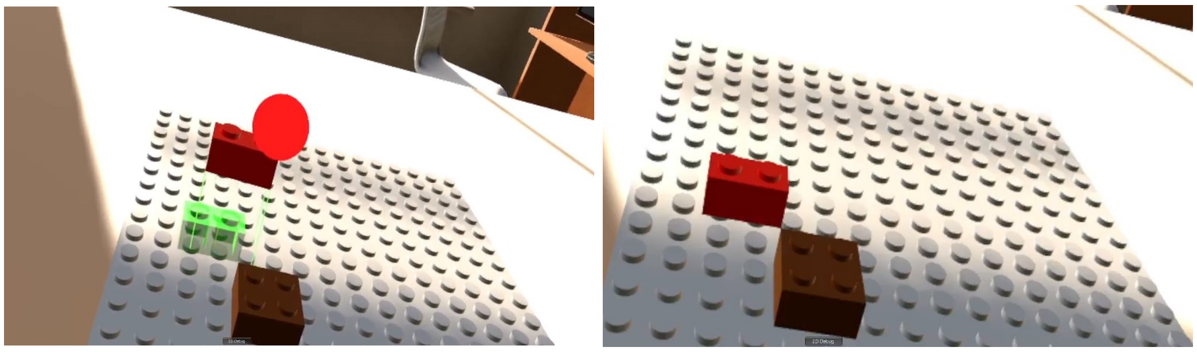

Figure 4: Placement estimation with floating bricks

To fix this newly detected issue, a function was added, which receives a list of grid locations and the number of the corresponding grid. It then checks, whether a brick placed at any of the given positions would have a connection to another brick, as this is needed for realistic placement. If this was not the case for a given brick, the estimation function would then try to find a suitable placement spot in a grid below the given one.

Another issue detected during the pilot studies, was the unsatisfactory snapping behavior if the distance between bricks and build plate was extremely small. The estimation would throw errors, or estimate a position not in line with the user's expectations. Turning off the physical interactions for any bricks, if they are currently handled, already improved the close proximity snapping by a lot. However, it is now possible for the user, to move a brick through the build plate, since the brick receives no collisions, while being handled. As soon as the brick and base plate overlapped, the placement estimation would start throwing errors since the raycast no longer provided a hit point and thus, no intersection between the ray and the base plate. As it is very easy for the user to accidentally move a brick through the base plate, it was decided that this behavior needed to be changed.
Since the user is able to place a brick from almost infinitely above the plate, it should therefore also be possible from below the plate, to optimally be in line with the user's mental model. To this end, the direction of the raycast was conditionally changed, such that it would always point towards the base plate and not downwards. Whilst this fixed the placement estimation for bricks held below the base plate, it did not eliminate the problems caused by bricks entering the base plate. Then, the placement estimation would still return an error, stating that no intersection point with the build plate could be found.

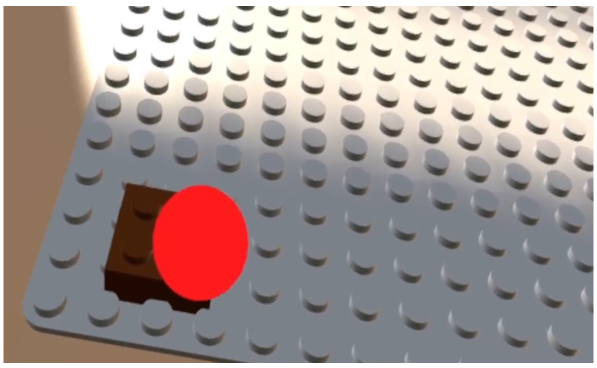

Figure 5: Brick inside plate with missing preview

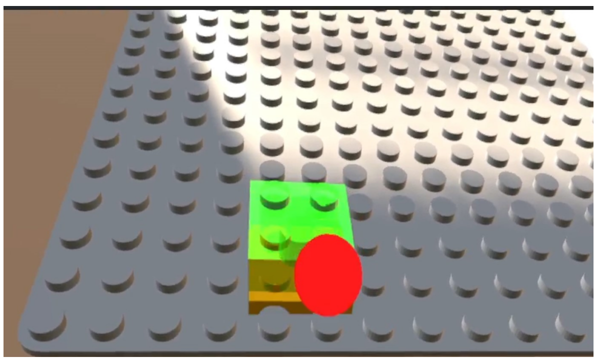

Figure 6: Preview inside plate with new system

This issue was caused by the inner workings of Unity's physics engine, which performs all the raycasts. This engine would only register intersection points with forward facing faces of mesh colliders, whilst backwards facing faces were ignored. Since the ray is cast from inside the base plate's collider, it would thus only hit backwards facing collider faces. Luckily, the intersection detection with backwards facing collider faces can be turned on by setting the "queriesHitBackFaces" flag within the physics engine. This is in the system's current state always turned on before a raycast and turned off afterwards, to prevent any unwanted bugs.

These changes made the placement estimation system much more intuitive and natural to use. This was especially noticeable when directly comparing the current version to the last iteration and constitutes a massive improvement in interaction quality.

As a means to further increase the software's usability, the preview's color was changed such that it can not be confused with a real brick (compare figures 1 and 3). The blue color that was used for the preview previously, lead to exactly these confusions during the pilot studies.
Lastly the verification of the software's quality and applicability for a new pilot study is in order. Therefore, a life demonstration shall be conducted in the near future.