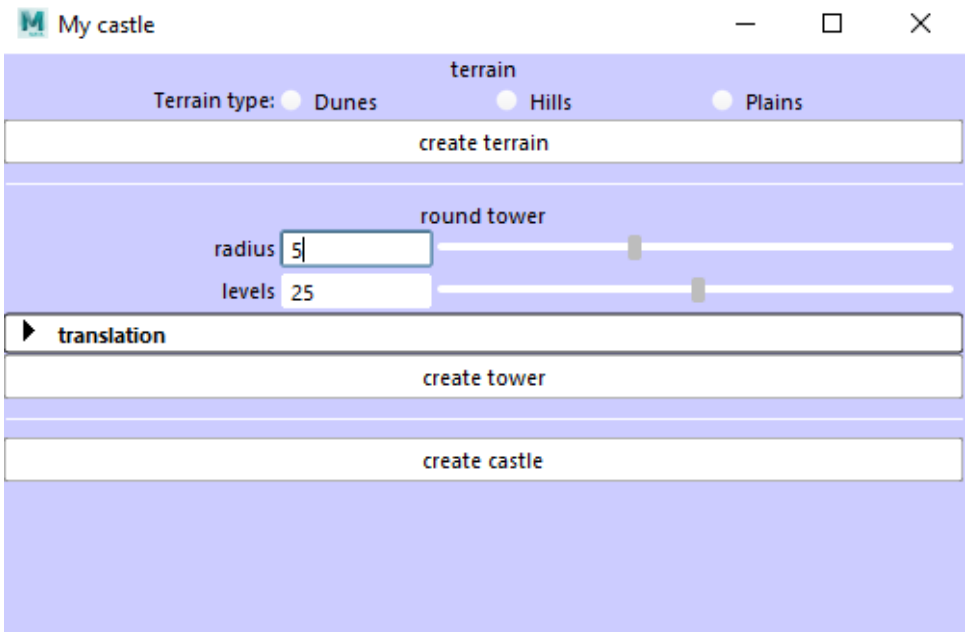
Castle Barrier – report

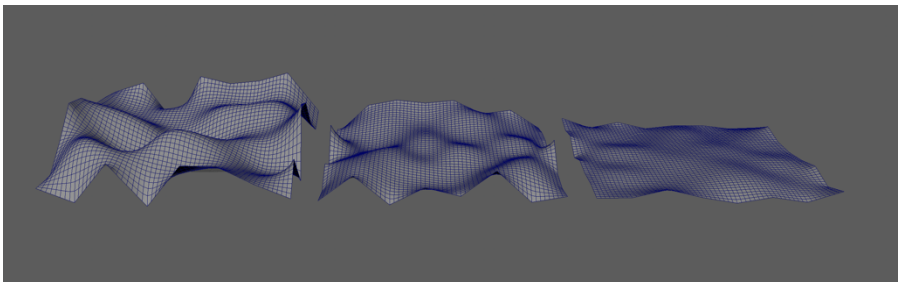
My code creates the outer part of a castle that is standing on top of a terrain. In my code I created a round building, a brick wall and a terrain that are being arranged based on the user's choice. The user is presented with a user interface window and can determine the level of mountainous for the terrain and can determine the attributes of the round buildings such as radius, height and location on the terrain (x-axis and z-axis). The program will already know to determine the y-position based on the area of the terrain the building is standing on. After that the user can press on the 'create castle' button and the brick walls will arrange themselves appropriately.

User manual



Section 1 – terrain:

- Choose the terrain type. Only one option can be chosen, and it is necessary to pick an option to proceed. (In the photo: Dunes, Hills and Plains, respectively)

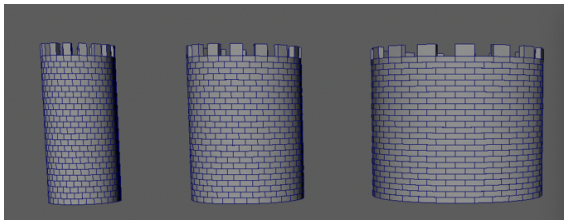


(Dunes, Hills and Plains, respectively)

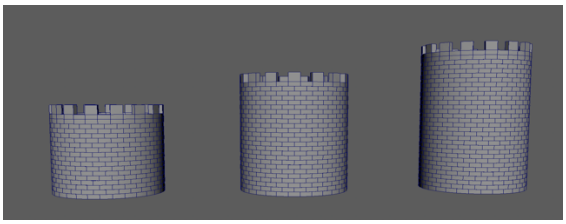
- 'Create terrain' button will create the chosen terrain when clicked.

Section 2 – round tower:

- Choose the radius and the height of the tower. The values shown in the picture are default.

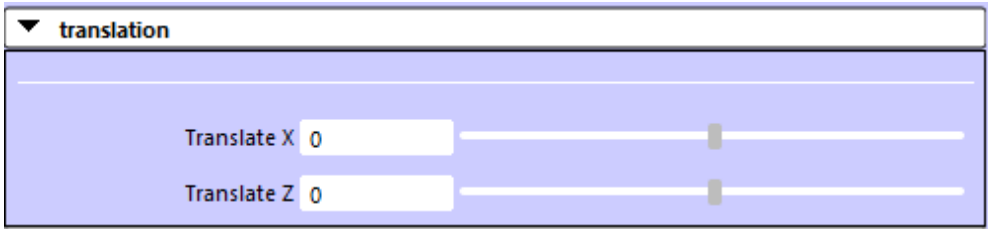


Towers with different radiuses

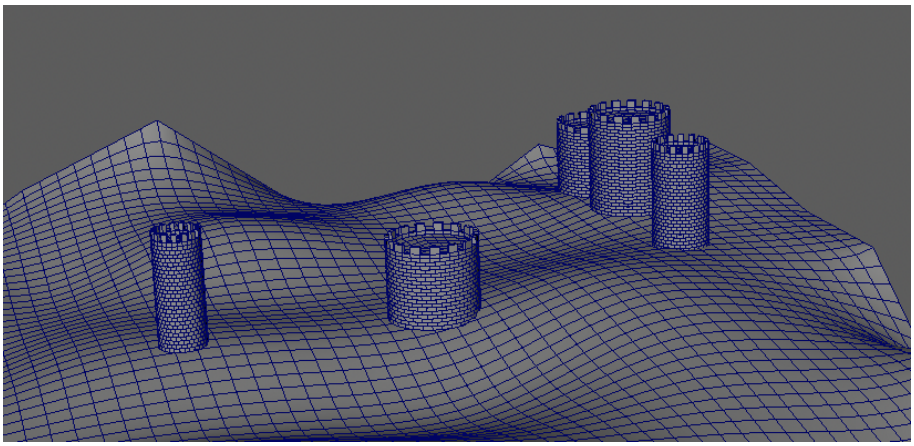


Towers with different heights

- Open translation menu by clocking on the black triangle. Select the translation attributes. (Y-axis translation will be determined by the height of the terrain at the location point of the tower).



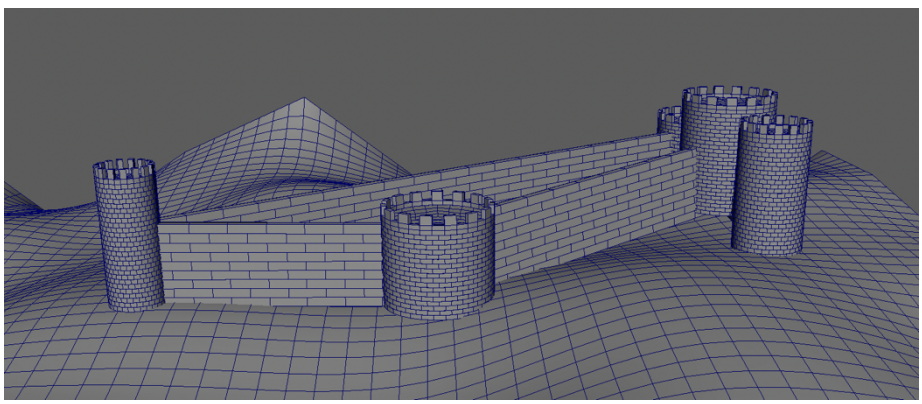
- 'Create tower' button will create a tower with chosen attributes when clicked. Must create at least 3 towers to proceed.



Towers arranged by the user on the terrain

Section 3 – create castle:

- 'Create castle' button will attach the towers with brick walls.
  Note: This process may take a few minutes (approx. 5)



The code functions

create_terrain

The function receives the type of terrain the user wants. It checks also if the user didn't choose any option which in that case a warning popup window will show, and no terrain will appear. The level of mountainous is influenced by the height attribute – the highest the value will be, the more mountainy it will be. The terrains' vertices are positioned randomly within their defined limits.

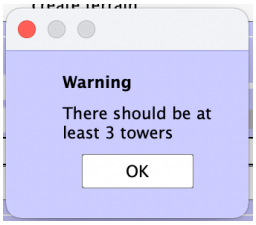
```
def create_terrain(*args):
    chosenTerrain = cmds.radioButtonGrp(terrainType, q=True, select=True)
    if chosenTerrain == 1:
        height = 50
    elif chosenTerrain == 2:
        height = 25
    elif chosenTerrain == 3:
        height = 10
    else:
    # in case when none of the options are selected
        cmds.confirmDialog(title='Warning', message='Please choose a terrain type', button='OK', defaultButton='OK', bgc=(0.8, 0.8, 1.0))
    width = 150
    depth = 150
    # Create a new polygon plane with the specified dimensions
    terrain = cmds.polyPlane(width=width, height=depth, subdivisionsWidth=7, subdivisionsHeight=7, name='terrain')
    if chosenTerrain == 0:
        cmds.delete(terrain)

    plane_name = cmds.ls(selection=True)[0]

    # Get the vertices of the polygon plane
    vertices = cmds.ls("{0}.vtx[*]".format(plane_name), flatten=True)

    # Set a random height value for each vertex
    for vertex in vertices:
        x, y, z = cmds.pointPosition(vertex, world=True)
        height_value = random.uniform(0, height)
        cmds.move(x, height_value, z, vertex, absolute=True)

    # Smooth the terrain to make it look more natural
    cmds.polySmooth(plane_name, dv=3)
```

Popup window will show up if no terrain type was selected

createCastle

The function checks if while clicking the button 'create castle' there are more than 3 towers (as there is not much to do with less than 3 towers). If there are, the function calls another function – *curves*, which I will explain later. If there are not enough towers, a warning popup window will show.

```
def createCastle(*args):
    cmds.select('finalTower*')
    selected_objects = cmds.ls(selection=True)
    if (len(selected_objects))/2<3:
        cmds.confirmDialog(title='Warning', message='There should be at least 3 towers', button='OK', defaultButton='OK', bgc=(0.8, 0.8, 1.0))
    else:
        curves(selected_objects)
```



Popup window will show up if there are not enough towers

createTower

The function receives the arguments from the GUI controls and then uses them to call two other functions – *verLocation* and *castle*, which will be explained later.

```
def createTower(*args):
    radius = cmds.intSliderGrp(radiusValue, query=True, value=True)
    levels = cmds.intSliderGrp(levelValue, query=True, value=True)
    translateX = cmds.intSliderGrp(positionX, query=True, value=True)
    translateZ = cmds.intSliderGrp(positionZ, query=True, value=True)
    translateY = verLocation(translateX, translateZ)
    castle(radius, levels, translateX, translateZ, translateY)
```

castle

The function builds the round tower. When called, it receives the radius, levels (height), and position attributes. The tower is created from polyPipe. In addition, as the ceiling, a pDisc is created and as the last level of the tower, the ‘crenellation’ is also made out of polyPipe but some faces are extruded upwards. Every level is made separately and at the end all levels are combined.

```
def castle(radius, levels, translateX, translateZ, translateY):
    numBricks = 15
    brickHeight = 1
    brickThickness = 0.5
    tower = cmds.polyPipe(sa=numBricks*2, h=brickHeight, r=radius, t=brickThickness, name="tower")
    for i in range (1, levels-1):
        newLevel=cmds.instance(tower)
        cmds.move(0,i*brickHeight/2, 0, newLevel)
        cmds.rotate(0, 90*i, 0, newLevel)
    top = cmds.polyPipe(sa=numBricks*2, h=brickHeight, r=radius, t=brickThickness, name="lastLevel")
    lastLevel = cmds.instance(top)
    cmds.delete(top)
    cmds.move(0, (levels-1)*brickHeight/2, 0, lastLevel)
    for x in range(numBricks*2, numBricks*4, 2):
        cmds.select(lastLevel[0] + ".f[%d]" % x, add=True)
        mySel = cmds.ls(selection=True, tail=1)
        cmds.select(clear=True)
        cmds.polyExtrudeFacet(mySel, ty=1)

    ceiling = cmds.polyDisc(r=radius-brickThickness)
    cmds.move(0, (levels-1.5)*brickHeight/2, 0, ceiling)
    cmds.polySoftEdge(ceiling, a=180, ch=0)
    cmds.select('tower*')
    towerGrp = cmds.group(name="towerGrp")
    cmds.select('pDisc*')
    cmds.group(parent='towerGrp')
    cmds.select('lastLevel*')
    cmds.group(parent='towerGrp')
    cmds.select('%s*' % towerGrp, replace=True)
    finalTower = cmds.polyUnite(ch=False, mergeUVSets=True, centerPivot=True, n='finalTower')
    cmds.delete(towerGrp)
    cmds.move(translateX, translateY, translateZ)
```

verLocation

The function searches for the closest vertex of the terrain to the position of the tower. As the user determines the attributes of z and x axis, it is not possible for him to determine the y-axis as it is changing depending to the terrain mountainous ‘ texture’. The function at the end returns the y attribute in order for the tower to ‘stand’ on the terrain.

```
def verLocation(translateX, translateZ):
    chosen_location = (translateX, 25, translateZ)

    # Get the vertices of the terrain
    vertices = cmds.ls(terrain_name + ".vtx[*]", flatten=True)

    # Set an initial minimum distance and closest vertex
    min_distance = float("inf")
    closest_vertex = None

    # Iterate over the vertices and find the closest one
    for vertex in vertices:
        vertex_pos = cmds.pointPosition(vertex, world=True)
        distance = math.sqrt(sum((v1 - v2) ** 2 for v1, v2 in zip(chosen_location, vertex_pos)))
        if distance < min_distance:
            min_distance = distance
            closest_vertex = vertex
        if closest_vertex:
            cmds.select(closest_vertex)
    selected_vertex = cmds.ls(selection=True)[0]

# Convert the selected vertex to a vertex component
    vertex_component = cmds.polyListComponentConversion(selected_vertex, fromVertex=True, toVertex=True)

# Retrieve the world space position of the vertex
    vertex_position = cmds.pointPosition(vertex_component[0], world=True)
    print("Vertex Position:", vertex_position[1])
    return vertex_position[1]
```

get_object_position

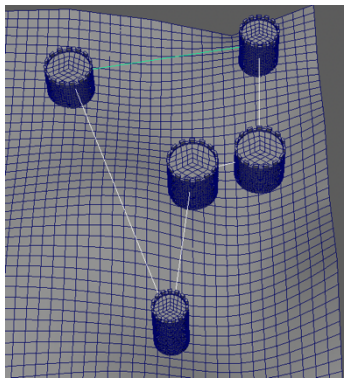
The function obtains the towers’ world space position.

```
def get_object_position(obj):
    try:
        return cmds.xform(obj, query=True, translation=True, worldSpace=True)
    except:
        return None
```

Curves

The function sorts all towers in the scene by their location. This is in order for them to be in an order that is not affected by the order of their creation because at the end, what we are building is a sort of a barrier and we need to connect all the towers with walls. After the towers are sorted in the list 'sorted_objects', they are listed again but this time only in pairs. This is in order to create a curve between two towers and not one curve that goes through all of them.

```
def curves(selected_objects):
    # Filter and remove None values from the list
    valid_object_names = [obj for obj in selected_objects if get_object_position(obj)]

    # Sort the objects based on their location
    sorted_objects = sorted(valid_object_names, key=get_object_position)

    first_object = sorted_objects[0]

    # Append the first object to the end of the list
    sorted_objects.append(first_object)

    for i in range(len(sorted_objects) - 1):
        object1 = sorted_objects[i]
        object2 = sorted_objects[i + 1]
        another = [object1, object2]
        print(another)

        object_positions = []
        for obj in another:
            position = cmds.xform(obj, query=True, translation=True, worldSpace=True)
            object_positions.append(position)
            print(object_positions)

        curve_name = cmds.curve(degree=1, point=object_positions)
        curve_length = cmds.arclen(curve_name)
        print(curve_name)
        print(curve_length)
        build_walls(curve_length, curve_name)
```

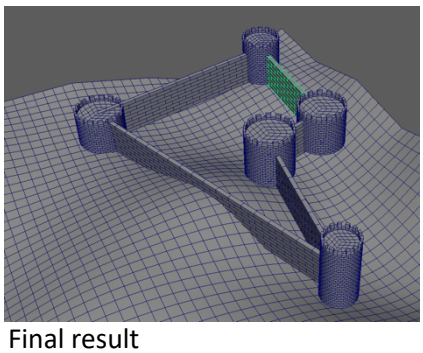


The curves connecting between two 'neighbor' towers

Build_walls

The function creates the brick walls. It builds brick by brick and at the end combines them. When there is a wall ready, we align the wall to the curves between the towers.

```
def build_walls(length, curve_name):
    print(length)
    brickWidth=4
    brickHeight=1
    brickDepth=1
    numRows=9
    numColumns=int(length/brickWidth)
    objName=cmds.polyCube(w=brickWidth,h=brickHeight,d=brickDepth,n='brick')
    halfObjName=cmds.polyCube(w=brickWidth/2.0,h=brickHeight,d=brickDepth,n='halfBrick')
    for y in range(numRows):
        if y%2==1: # 1%2=1; 2%2=0; 3%2=1;4%2=0;5%2=1;6%2=0
            offset=brickWidth/2.0
        else:
            offset=0.0
        for x in range(numColumns):
            if x==0 and y%2==0:
                newObj = cmds.instance(halfObjName)
                xpos = x*brickWidth+offset+brickWidth/4.0
            elif x==numColumns-1 and y%2==1:
                newObj = cmds.instance(halfObjName)
                xpos = x*brickWidth+offset+brickWidth/4.0-brickWidth/2.0
            else:
                newObj = cmds.instance(objName)
                xpos = x*brickWidth+offset
            cmds.move(xpos,y*brickHeight,0,newObj)
    cmds.delete(objName)
    cmds.delete(halfObjName)
    cmds.select(clear=True)
    selected = cmds.ls("brick*")
    selected.append(cmds.ls("halfBrick*"))
    for i in selected:
        cmds.select(i, add=True)
    Grp = cmds.group(name='wall')

    # Select all objects in the group
    cmds.select('%s*' % Grp, replace=True)

    # Combine the objects into a single object
    united = cmds.polyUnite(ch=False, mergeUVSets=True, centerPivot=True, n='united')
    cmds.delete(Grp)
    cmds.select(clear=True)
    warp_deformer = cmds.createCurveWarp(united, curve_name)
```



Final result

Main function

Specifies all the GUI controls and attributes that will be shown in the user interface window.

```
import maya.cmds as cmds
import math as m

window = cmds.window(title="My castle", widthHeight=(500, 500), bgc=(0.8, 0.8, 1.0))

cmds.columnLayout(adjustableColumn=True)
cmds.text('terrain')
terrainType = cmds.radioButtonGrp( label='Terrain type:', la3=['Dunes', 'Hills', 'Plains'], numberOfRadioButtons=3)
cmds.button(label='create terrain', command=create_terrain, bgc=(1.0, 1.0, 1.0))
cmds.separator(height=20)
cmds.text('round tower')
radiusValue = cmds.intSliderGrp(field=True, label='radius', minValue=2, maxValue=10, value=5)
levelValue = cmds.intSliderGrp(field=True, label='levels', minValue=15, maxValue=35, value=25)
frame_layout = cmds.frameLayout(label='translation', collapsable=True, collapse=True, borderVisible=True, marginWidth=5, marginHeight=5, bgc=(1.0, 1.0, 1.0))

cmds.separator(height=20)
positionX = cmds.intSliderGrp(field=True, label='Translate X', minValue=-60, maxValue=60, value=0)
positionZ = cmds.intSliderGrp(field=True, label='Translate Z', minValue=-60, maxValue=60, value=0)
# Close the frameLayout
cmds.setParent("..")
cmds.button(label='create tower', command=createTower, bgc=(1.0, 1.0, 1.0))
cmds.separator(height=20)
cmds.button(label='create castle', command=createCastle, bgc=(1.0, 1.0, 1.0))
cmds.showWindow(window)
```

References

- In ‘castle’ function, I took the idea on how to build the tower from the slides in brightSpace: Gears Systems p.13
- In ‘build_walls’ function, I took the idea from brighSpace slides. Introduction to production tool->Functions->7_wall
- In ‘build_walls’ function, I took the code that was in the forum there. I was looking for a way to use warp deformer with python. (MASH1_ReproMesh, $selection[0]); - I changed the line for it to apply on python and to fit my needs. https://forums.autodesk.com/t5/maya-programming/scripting-curve-warp/td-p/9964799
- Wathed this video that helped me understand Maya GUI. https://www.google.com/search?q=maya+python+gui+examples&oq=maya+python+gui&aqs=chrome.0.69i59l2j69i57j0i22i30l2j69i60l3.4686j0j7&sourceid=chrome&ie=UTF-8#fpstate=ive&vld=cid:1afeb39c,vid:n4i4F2fmK2M
- Autodesk website https://download.autodesk.com/us/maya/2010help/commandspython/cat_Windows.html
- In ‘verLocation’ function, lines 126-134, I found some codes on how to find coordinates of the closest point and took inspiration and reformed them a bit to fit my needs. https://stackoverflow.com/questions/24415806/coordinates-of-the-closest-points-of-two-geometries-in-shapely