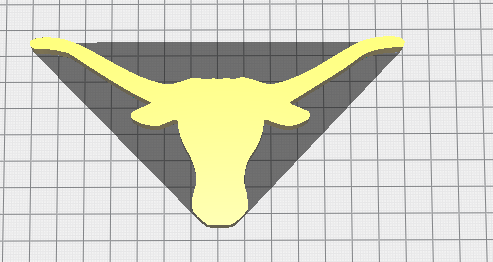
Chunking and Placement Physical Validation Process

1. Create Model: In this case, we use a model of the Texas Longhorns logo by plexaterson (<https://www.thingiverse.com/thing:1200275>), resized to 630mm x 320mm x 45mm tall. The original file as well as well as the resized version can be found in the “STL Files” folder. This size was chosen because it creates multiple chunks. 
2. Determine how many layers we should create. This should be no more than the number of mobile transport robots because in our MSEC paper, we assume that there will always be a mobile robot available to move each printing robot. We will arbitrarily use layer height of 15mm to facilitate the creation of multiple chunks while keeping print times down. This results in 3 layers, which is the number of mobile platform robots we have
3. The next step is z-chunking. To do this use the “Assembly Geometry” branch of the scaled chunker repo on bitbucket. I will save locally the exact settings I used to create this model in the “Z-Chunking Code” folder included with these instructions. You will need to manually adjust the settings for “max\_reach\_z”, which controls the height of the printer and will be artificially lowered to allow for a layer height of 15mm, and “vertical\_check\_density”, which determines the density of possible chunking locations, for future use in ”Z-Chunking Code\am3\chunker.py” (For more info on these parameters see *Z-Chunking for Cooperative 3D Printing* in the SFF 2022 archives. To run this code:
   1. Open the command prompt in windows
   2. Navigate to the “Z-Chunking Code” folder with command *cd [folder address (you can just drag and drop this from the file browser)]*
   3. Run the following command: *[blender.exe address] -b -P [blender\_chunk.py address] -- -f [STL file address] -n 3 -c*
   4. There should now be the file for each layer (job) with added AG (Assembly geometry) in the same location your STL file was saved. In this case I have saved the layers in “STL Files\Z-Chunking Files” folder labeled 10 (for layer 1) to 30 (for layer 3) from the bottom up.
4. Next is XY chunking. For this use the “command line chunking” branch of the scaled chunker repo on bitbucket. Again, I will save the exact code I used in the “XY Chunking Code” folder. The instructions for this are slightly different and you will follow these for each individual job. The reason this is different is because there are some bugs with how XY chunking and Z chunking work with each other, so I have removed XY chunking code from the Z-Chunking folder.
   1. Open the command prompt in windows
   2. Navigate to the “Z-Chunking Code” folder with command *cd [folder address (you can just drag and drop this from the file browser)]*
   3. Run the following command: *[blender.exe address] -b -P [blender\_chunk.py address] – [STL file address]*
   4. The individual chunks should now be in the “exports” folder within the “XY Chunking Code” folder
5. Next, we need to figure out the dependencies. The XY Chunking code should do this automatically, but I haven’t found where it is listed in this old version of the software. It can be easily done manually on a per-layer basis by looking at the STL files. The chunk labeled “0” in each layer will always be the initial chunk with no dependencies. After that, you should re-assemble the chunks and look at the slope relationships in the layer. Chunks with no overhanging slope are independent. Chunks with overhanging slopes are dependent on the chunk in the direction of each overhang. Use this information to build the dependency tree as described in (A Generative Approach for Scheduling Multi-Robot Cooperative Three-Dimensional Printing).
6. Calculate the print times in Cura. You do this by creating a custom printer that is slightly bigger than the job in multiples of buildplate size (in this case 900mm by 600mm). Then, import each chunk individually and chunk to a .45 layer height resolution and record the print time in minutes for each chunk of each job.
7. Next, we need to run the placement optimization code. For this you need to input the chunk dependencies, which job each chunk belongs to, and the chunk print times into the “placer.py” file in the “\Placement Codes\PythonCBS\All Codes” folder. The example format as well as the exact code I used for this will be in this file. It should be noted that chunks are referred to by different numbers in this process. They are numbered sequentially by row starting from the bottom chunk. For the longhorn logo example there is only one row, so the chunks in layer 0 will be “0, 1, and 2”, the chunks in layer 1 will be “3, 4, and 5” and so on. If any layer had more than one row, the order would be layer 0 row 0 -> layer 1 row 0 -> layer 2 row 0 -> layer 0 row 1 -> layer 1 row 1 -> layer 2 row 1 and so on.
8. When run, the placer code will optimize the placement to minimize makespan. The results will be in a text file called “…\All Codes\GA Results\Results.txt”. At the bottom will be the final makespan, schedule for robots 0 to i, and chunk positions in order from 0 to n for each job from the bottom up. There will also be additional files “Configurations” and “Print\_Times” in this folder which track the progression of the genetic algorithm over time.
9. Print each chunk in the correct schedule for each robot with the corresponding positions. To do this you will need to use the ambots app, create a 1x1 build area for each location, upload the STL file for that chunk, and then print. You will likely need to calibrate at each position and may need to shift the chunks to align with eachother. Record the print time and move time for each step (but calibration time can be left out). The robots should automatically record the print time in the web UI.
10. Add up all print and move times for each robot and compare the longest of those to the makespan predicted by the algorithm.

Software Versions Used

* Python 3.7.9
* Blender 2.79b
* Cura 4.13.1

Repositories

* AMBOTS main bitbucket: Ask Dr. Zhou for link
  + Scaled-chunker is the chunking repository with branches for Z-Chunking (Assembly-Geometry) and XY Chunking (command-line-chunking)
* Placement Repository: <https://github.com/DanWebUT/Placement>
  + This has all of the work and progress on the placement algorithm
* Layer-Based-C3DP: <https://github.com/DanWebUT/Layer-Based-C3DP>
  + This has the work I have done on the slicer for the Layer Based Project with TAMY
* Chunking and Placement: <https://github.com/DanWebUT/Chunking-and-Placement>
  + The repository where these instructions and code are located explaining how to physically carry out the C3DP of a large and tall object.

Current issues with mobile platform

The main hurdle in the physical validation so far has been that the mobile platform robots do not consistently work. The primary issue is that they have trouble reading the QR codes on the floor and therefore cannot move anywhere (because they need to check the QR code each time after moving one space). Symptoms are that the robot will move in the requested direction, but then just wiggles around at the destination location. From this, I have concluded that three the color sensor which keep the robot moving along the black lines are working correctly, but that the camera is the issue. In brief, preliminary testing, this issue was shared across all three mobile platforms. I have taken one robot apart to visually check for damage to the camera and ribbon cable and none were found. Here are some next troubleshooting steps to try:

1. Turn everything (robots, floor, and control hub) off and back on and see if things magically get fixed.
2. Ensure the robots are fully charged.
3. Clean the camera lenses thoroughly
4. Use a multimeter to check the ribbon cable

Beyond that, you should reach out to Zac Hyden for help. He had some experiments to try on the software side of things that I did not have time to validate.