ChatGPT

Adding CNC Lathe Toolpath Simulation to IntuiCAM

Existing CNC Simulation Libraries (Lathe Support)

Before implementing from scratch, it's worth surveying available CNC simulation tools. **CAMotics** (open-source) is a popular G-code simulator, but it currently supports only 3-axis milling and explicitly has *no lathe simulation* as of its latest release [1]. This means you would need to extend CAMotics significantly to handle turning. Another option is the **CNC Web Simulator** by Filipe Caixeta, an open-source browser-based simulator (MIT-licensed) that can simulate mills, 3D printers, *and lathes*. It uses WebGL for visualization and provides lathe toolpath and "2D simulation" with a generated 3D model [2]. This suggests it performs the lathe stock removal in a 2D cross-section view and then revolves the result to show a 3D workpiece. While its code is in JavaScript/WebGL, it can serve as a reference for how to structure a lathe simulation (e.g. using a 2D depth-buffer approach for material removal and then rendering a revolved solid).

On the commercial side, there are high-end SDKs like **MachineWorks** or **ModuleWorks**, which are toolkits offering fast material removal simulation, collision detection, and full machine kinematics for all CNC types (including lathes) [3] [4]. These are well-proven but are proprietary (costly and closed-source). They use optimized algorithms (often voxel/dexel hybrid models and GPU acceleration) to achieve real-time performance [4]. If open-source options fall short and budget allows, integrating a commercial SDK is an option – but given your preference for open source and the C++/Qt environment, a self-implemented or open library solution is preferable.

Another relevant open project was **HeeksCNC**, an open-source CAM program (C++ with OpenCascade) which integrated a volumetric simulation called *VoxelCut* for material removal [5]. Dan Heeks's approach converted the stock into a voxel/octree data structure and subtracted the tool volume incrementally. This achieved a **"solid simulation"** of milling operations [5]. While the HeeksCNC code (on GitHub) might provide insights, note that it was primarily for milling; lathe simulation specifics would still need development. In summary, no turnkey open-source library exists *specifically* for CNC lathe stock removal in C++ at this time – so you will likely be implementing it yourself, drawing on known techniques from literature and other simulators.

Real-Time Material Removal Techniques

For real-time visual simulation of material removal, using naive CAD boolean operations for each tool motion is **too slow**. As OpenCascade's own experts note, its boolean cuts are *"not dedicated to be used in real time"* for continuous material removal [6]. Instead, simulation systems use specialized data structures to represent the stock dynamically. The three primary approaches are **voxels**, **dexels**, or **dynamic meshes**, each with trade-offs:

- **Voxel Grids (3D):** The stock is represented as a 3D array of tiny cells (voxels). When the tool cuts the material, the corresponding voxels are cleared. This method is simple to implement and works for any geometry, but high resolution voxel grids can consume a lot of memory. Octree or adaptive grids (subdividing only areas near detail) can optimize this [7]. Dan Heeks's VoxelCut was essentially a voxel/octree approach for milling simulation. A voxel model can be rendered by extracting a mesh (e.g. via Marching Cubes, as Anders Wallin did with an octree + marching cubes to display the surface [8]). Voxels can achieve real-time rates if the resolution is tuned and updates are localized, but for fine lathe detail you'd need a fairly high resolution in two axes (radial and axial).

- **Dexel (Depth-Element) Model (2.5D):** Dexel simulation is very popular for CNC because it is much more memory- and time-efficient for many cases. A *dexel* is essentially a vertical column or ray that stores the depth of material along that line. For a **3-axis mill**, you can imagine a grid of columns in the XY plane: each column stores the height of remaining material (Z extent). For a **lathe (2-axis turning)**, you can use a *cylindrical dexel model*: a set of radial lines around the stock or a 2D grid in the radial (R) and axial (Z) directions. As the cutting tool moves, you update the intersection of the tool with these lines – effectively carving away the material in the cross-section profile. This is essentially what the web simulator above does by providing a "2D simulation" for lathe [2]. Because a lathe's result is axisymmetric, a 2D cross-section (R–Z plane) is enough to represent the workpiece shape. You update the cross-sectional outline as the tool removes material, then revolve it for 3D visualization. The dexel approach is known to be **fast** and suitable for real-time: as one researcher noted, *"Dexel (or LDI) is the best solution out of Octree, triangle, or voxel simulation"* for speed and accuracy [9]. In practice, you'd allocate a grid (Z vs radius or Z vs X if working in diameter) with a resolution fine enough for your tolerance, and at each tool movement, compute which grid cells the tool sweeps through to shorten those dexels. This can be hardware-accelerated (there are papers on GPU dexel simulation for cutting). The classic reference is Van Hook's algorithm (1980s) which introduced dexels for milling; for turning, it's even more straightforward since the tool sweep is 2D.

- **Dynamic Mesh Boolean:** Another approach is to represent the stock as a polygonal mesh (for example, a finely tessellated cylinder) and perform Boolean subtractions of the tool shape volume from this mesh as the tool moves. This can be done via mesh boolean libraries or custom algorithms. While potentially more geometric-accurate, this tends to be slower than dexels/voxels, especially if using a robust CAD kernel for each cut. Some simulation systems use a hybrid: e.g. maintain a mesh for display but use a voxel/dexel model under the hood for fast updates, only occasionally updating the mesh from the volumetric model for visualization. Given you already use OpenCASCADE (OCCT) in IntuiCAM, you might be tempted to use OCCT's Boolean operations to cut the stock. **For real-time continuous simulation, however, this will not be feasible** – OCCT booleans could maybe recompute a final shape after an operation, but doing it for every few mm of tool motion would bog down. OCCT could still be useful to generate an accurate final part geometry at the end (e.g. by revolving the final profile or doing one final cut), but not for the frame-by-frame removal simulation [6].

In summary, the **recommended technique for a lathe** is to use a **2D cross-sectional simulation (dexel or even pixel-based)**. For example, represent the stock's cross-section as an array of radial depths at various Z positions. As the tool moves and cuts, compute the engagement in that cross-section (the tool can be represented by its 2D profile shape) and remove material by reducing the radius values. This is essentially a rasterization/clipping operation in 2D, which is extremely fast. Once updated, you can display the new cross-section or revolve it to update a 3D model view. This approach is memory-efficient and fast enough for real-time, since you're only updating a 2D grid. Academic projects have even done this by "sweeping" the tool profile and using polygon clipping to subtract it from the stock profile polygon [10] [11], which is another way to get the updated cross-section accurately.

Implementation in C++ with Qt6 and OCCT

Given IntuiCAM is C++/Qt6/OpenCASCADE, you have a few integration options:

- **Use OpenCASCADE for Visualization:** You can continue using OCCT's visualization (AIS shapes) or Qt's 3D for displaying the stock and tool. For instance, you might create an OCCT shape for the initial stock (a cylinder) and for the cutting tool (perhaps a simple shape like a profile revolved, or even just use its profile for 2D simulation). During simulation, if using a dexel model, you won't continuously boolean the OCCT shapes. Instead, you'll update your dexel data structure and periodically update a mesh or shape for display. One strategy is to generate a triangulated mesh of the stock from the dexel grid and then display that mesh in the viewer. OCCT can create a mesh from points, or you could use vtk/Qt 3D to render the point cloud or height map. CAMotics, for example, allows exporting the simulated workpiece to an STL mesh [12], hinting that internally it maintains a mesh or voxel model that can be output. You could similarly keep an **internal model (grid)** and only convert to an OCCT shape occasionally (since converting a large mesh to an OCCT solid might be slow). Alternatively, skip OCCT for the stock display and draw the mesh directly with an OpenGL widget in Qt for performance.

- **Leverage OpenGL/GPU:** Since you want real-time visual feedback, GPUs can be extremely helpful. A technique often used is treating the removal simulation like a rendering problem: for instance, projecting the tool onto a buffer. In a lathe, you could imagine an offscreen 2D buffer where pixel values represent the current radius at that Z (almost like an image of the cross-section). Each tool pass can be rasterized into that buffer (deducting material). In fact, the *dexel* idea is analogous to a Z-buffer render from one side. Some research projects implemented dexel simulations on GPU by casting many rays (dexels) and updating them in parallel [9]. If you're comfortable with OpenGL/GLSL, you could encode the stock as a texture (depth map) and have a fragment shader subtract tool shapes very quickly. Otherwise, a CPU implementation with a 2D array updated in loops can still achieve interactive speeds, especially in C++.

- **Tool and Holder Modeling:** Don't forget to model not just the cutting edge but the tool body/holder if needed for collision checks. For a lathe, the cutting insert has a certain shape (usually a triangle, diamond, etc in 2D profile) and the holder might stick out. In simulation, usually the material removal is computed from the **cutting portion** only, but for crash detection you should also check if non-cutting parts of the tool or holder intersect the stock. With a dexel grid, an easy check is to detect if any dexel that should remain (material) is being penetrated by the tool outside of intended cutting contact – essentially if the tool moves into material too deep or at a bad angle. You might supplement this by having a simplified mesh or shape for the tool holder and use OCCT's collision algorithms to test against the stock shape periodically. However, most lathe crashes (apart from code errors) come from either tool over-engagement or the tool hitting the chuck/fixtures. If you plan to simulate the machine envelope too, you'd need 3D collision models for those components and check them. This might be beyond scope initially, so focusing on stock-tool collision is a good start (the simulation inherently covers that by showing any unexpected material removal).

Collision and Verification Considerations

One of the goals is to **detect crashes or collisions**. A properly done material removal simulation already helps reveal if the toolpath goes through more material than intended (e.g. a gouge or a collision with remaining stock). If using a volumetric model, you can flag a "crash" if the tool is commanded to remove an impossibly large volume instantaneously or if it tries to cut outside the stock's boundary in one move (indicating the tool would hit uncut material aggressively). Some simulators also check for *toolholder collisions*, i.e., if the holder or non-cutting part of the tool intersects the stock or other geometry. This requires modeling the whole tool assembly. In an open-source context, you would likely have to implement this geometric check yourself (for example, by sampling the tool geometry against the stock grid). Since IntuiCAM already uses OCCT, you could use OCCT's collision detection module on simplified shapes: represent the stock as a mesh or cloud, the tool holder as a polyshape, and check for intersections. Keep in mind OCCT collision checks might also be heavy in real-time; a simpler approach is to inflate the dexel removal for the cutting edge by the holder clearance and see if any material remains in that zone.

For verification, it's also useful to compare the final simulated part vs. the intended design. If you have a CAD model of the desired final shape, you can compare it to the simulated stock (e.g., via a mesh comparison or point-cloud distance check) to detect overcuts or undercuts. This goes beyond crash detection into quality verification, but could be a later extension.

In summary, to add lathe simulation to IntuiCAM: an **open-source friendly approach** is to implement a dexel-based or cross-sectional simulation engine in C++. Use a 2D array to represent the radial stock profile, update it as G-code commands are processed, and render the results (update an OCCT shape or a custom mesh in Qt's view). The community and literature suggest this method will be fast enough for real-time visualization [9], unlike per-cut CAD booleans [6]. You can draw inspiration from existing tools (CAMotics for overall structure, CNC Web Simulator for lathe-specific handling, HeeksCNC's VoxelCut for volumetric logic). Starting with open-source algorithms ensures you can debug and tailor the simulation to your needs. Over time, if higher fidelity is needed, you could incorporate GPU acceleration or even consider hooking in a proven engine (commercial or otherwise), but the dexel approach should meet the needs of interactive crash-checking and visualization for a 2-axis lathe.

Sources: CAMotics documentation on features/limitations [1]; HeeksCNC project notes [5]; OpenCascade forum discussion on simulation performance [6] [9]; CNC Web Simulator project README [2].

---

[1] [12] CAMotics
https://camotics.org/

[2] GitHub - filipecaixeta/cncwebsim: A cnc web simulator
https://github.com/filipecaixeta/cncwebsim/

[3] MachineWorks - MachineWorks Software Develoment Toolkit for ...
https://www.machineworks.com/

[4] MachineWorks 8.6 New Features
https://www.machineworks.com/blog/machineworks-8.6-new-features/

[5] HeeksCAD & HeeksCNC - Details
https://sites.google.com/site/heekscad/home/Details

[6] [9] CNC simulation - Forum Open Cascade Technology
https://dev.opencascade.org/content/cnc-simulation

[7] [Emc-users] Is there an open source program similar to Vericut?
https://emc-users.narkive.com/CoOP0yZo/is-there-an-open-source-program-similar-to-vericut

[8] CAM – anderswallin.net
http://www.anderswallin.net/cam/

10 11 Microsoft Word - Emre Yegin MSc Tezi - v1.0.docx
https://open.metu.edu.tr/bitstream/handle/11511/20218/index.pdf?sequence=1