Inverse and Forward Kinematics for a Delta Printer

Guadalupe Bernal
guadabernal@gmail.com

ABSTRACT

Inverse and forward kinematics for a delta printer

CCS CONCEPTS

• **Computing methodologies** → **Parallel programming languages**; **Computer vision**; *Image processing*;

KEYWORDS

OpenCL, OpenVX, Interoperability, Computer Vision.

ACM Reference format:
Guadalupe Bernal. 2017. Inverse and Forward Kinematics for a Delta Printer. In *Proceedings of IWOCL '17, Toronto, Canada, May 16-18, 2017,* 1 pages.
DOI: http://dx.doi.org/10.1145/3078155.3078183

1 INTRODUCTION

2 INVERSE KINEMATICS

The delta printer is modeled by three columns A, B and C, each one positioned at a vertex in an equilateral triangle. We define the point $\boldsymbol{q} \in \mathbb{R}^3$ $\boldsymbol{q} = [q_x, q_y, q_z]'$ as the point of extrusion with respect to the bed. The effector plane lies above $\boldsymbol{q}$ by a distance h. The distance between the effector plane and the joint at a carriage is defined by A_c, B_c, C_c The height of the carriages with respect to the bed is defined by A_z, B_z, C_z. Then

$$\begin{aligned} A_z &= q_z + A_c + h \\ B_z &= q_z + B_c + h \\ C_z &= q_z + C_c + h \end{aligned} \tag{1}$$

The length of a the rods is defined by ℓ so The distances between the effector joints and the carriages is defined by AD, BD, CD

$$\begin{aligned} \ell^2 &= {A_c}^2 + {A_d}^2 \\ \ell^2 &= {B_c}^2 + {B_d}^2 \\ \ell^2 &= {C_c}^2 + {C_d}^2 \end{aligned} \tag{2}$$

3 FORWARD KINEMATICS

4 SOFTWARE IMPLEMENTATION

Listing 1: OpenCL Interop-kernel

```
// Get OpenCL context associated with an OpenVX target
cl_context clContext = vxGetOpenCLContext(vxContext, targt);
// OpenCL standard code for creating kernels
cl_program clProgram = clCreateProgramWithSource(&src, ...);
cl_kernel clKernel0 = clCreateKernel(clProgram, "k0", ...);
cl_kernel clKernel1 = clCreateKernel(clProgram, "k1", ...);
...
// Create an OpenVX kernel for OpenCL interop
vx_kernel vxKernel = vxAddOpenCLInteropKernel(targets, ...,
userFunc, userVal, userInit, userDeinit);
// Attach OpenCL kernels to an OpenVX interop kernel
vxAddOpenCLKernelToKernel(vxKernel, 0, clKernel0);
vxAddOpenCLKernelToKernel(vxKernel, 1, clKernel1);
// OpenVX standard code for user-kernels
vxAddParameterToKernel(vxKernel, 0, VX_INPUT, ...);
vxAddParameterToKernel(vxKernel, 1, VX_OUTPUT, ...);
vxFinalizeKernel(vxKernel);
vx_node node = vxCreateGenericNode(graph, kernel);
vxSetParameterByIndex(node, 0, inputImage);
vxSetParameterByIndex(node, 1, outputImage);
vxProcessGraph(graph);
```

5 ERROR ANALYSIS

6 EXPERIMENTATION

7 CONCLUSIONS

REFERENCES

[1] S. Gauthami. 2017. The Khronos OpenVX 1.1 Specification. The Khronos OpenVX Working Group. (2017).
[2] A. Munshi. 2012. OpenCL 1.2 Specification. The Khronos OpenCL Working Group. (2012). http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf
[3] Erik Rainey, Jesse Villarreal, Goksel Dedeoglu, Kari Pulli, Thierry Lepley, and Frank Brill. 2014. Addressing System-Level Optimization with OpenVX Graphs. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW '14).* IEEE Computer Society, Washington, DC, USA, 658–663. DOI: https://doi.org/10.1109/CVPRW.2014.100

IWOCL '17, Toronto, Canada

DOI: http://dx.doi.org/10.1145/3078155.3078183