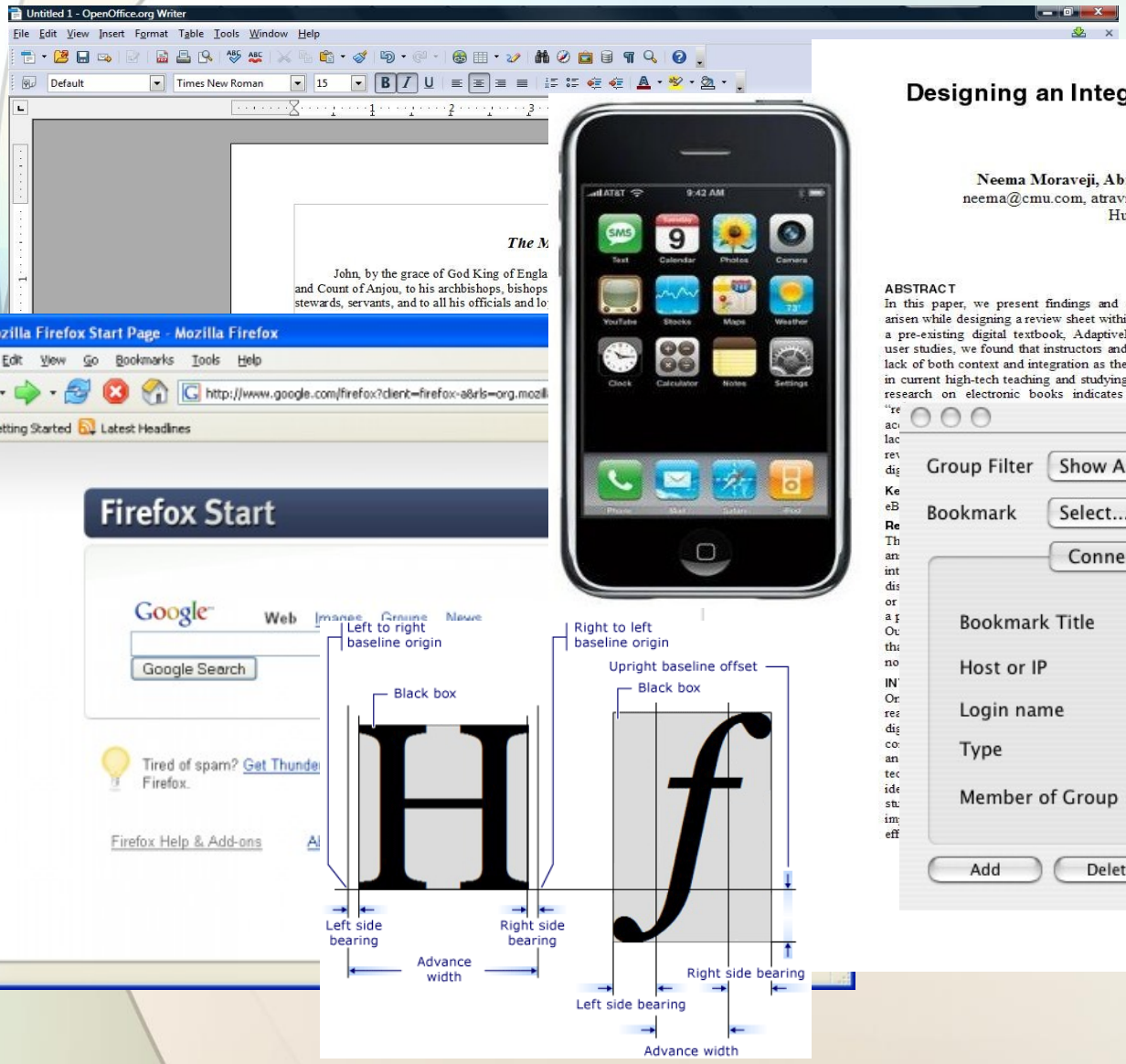
# The Nile Programming Language: Declarative Stream Processing for Media Applications

Dan Amelang
Viewpoints Research Institute
UCSD PhD Student

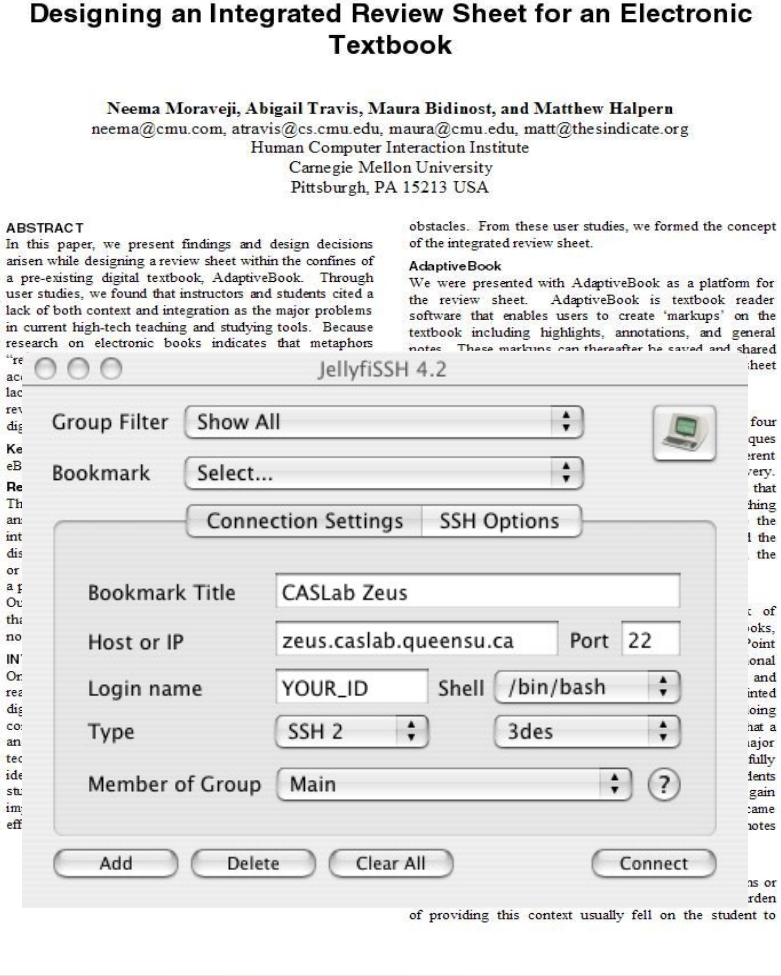
*Southern California Programming Languages and Systems Workshop – Spring 2011*

# 2D Vector Graphics









**Designing an Integrated Review Sheet for an Electronic Textbook**

**Neema Moraveji, Abigail Travis, Maura Bidinost, and Matthew Halpern**
neema@cmu.com, atravis@cs.cmu.edu, maura@cmu.edu, matt@thesindicate.org
Human Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213 USA

**ABSTRACT**
In this paper, we present findings and design decisions arisen while designing a review sheet within the confines of a pre-existing digital textbook, AdaptiveBook. Through user studies, we found that instructors and students cited a lack of both context and integration as the major problems in current high-tech teaching and studying tools. Because research on electronic books indicates that metaphors

obstacles. From these user studies, we formed the concept of the integrated review sheet.

**AdaptiveBook**
We were presented with AdaptiveBook as a platform for the review sheet. AdaptiveBook is textbook reader software that enables users to create 'markups' on the textbook including highlights, annotations, and general




of providing this context usually fell on the student to

# State of the Art – 10,000+ lines of...

Cairo (Firefox, Linux)

```
...
if (lxi > fill_start)
 {
      ADD_SATURATE_8 (ap + fill_start,
                      fill_size * N_X_FRAC (8),
                      lxi - fill_start);
       fill_start = lxi;
}
else if (lxi < fill_start)
{
      ADD_SATURATE_8 (ap + lxi,
                                N_X_FRAC (8),
                                fill_start - lxi);
}
if (rxi < fill_end)
{
       ADD_SATURATE_8 (ap + rxi,
                       fill_size * N_X_FRAC (8),
                       fill_end – rxi);
...
```

Skia (Chrome, Android)

```
...
if (currE->fLastY == curr_y) {
      if (currE->fCurveCount < 0) {
            if (((SkCubicEdge*)currE)->updateCubic())
      {
        SkASSERT(currE->fFirstY == curr_y + 1);
        newX = currE->fX;
        goto NEXT_X;
       }
  } else if (currE->fCurveCount > 0) {
  if (((SkQuadraticEdge*)currE)->updateQuadratic())
 {
    newX = currE->fX;
    goto NEXT_X;
   }
 }
...
```

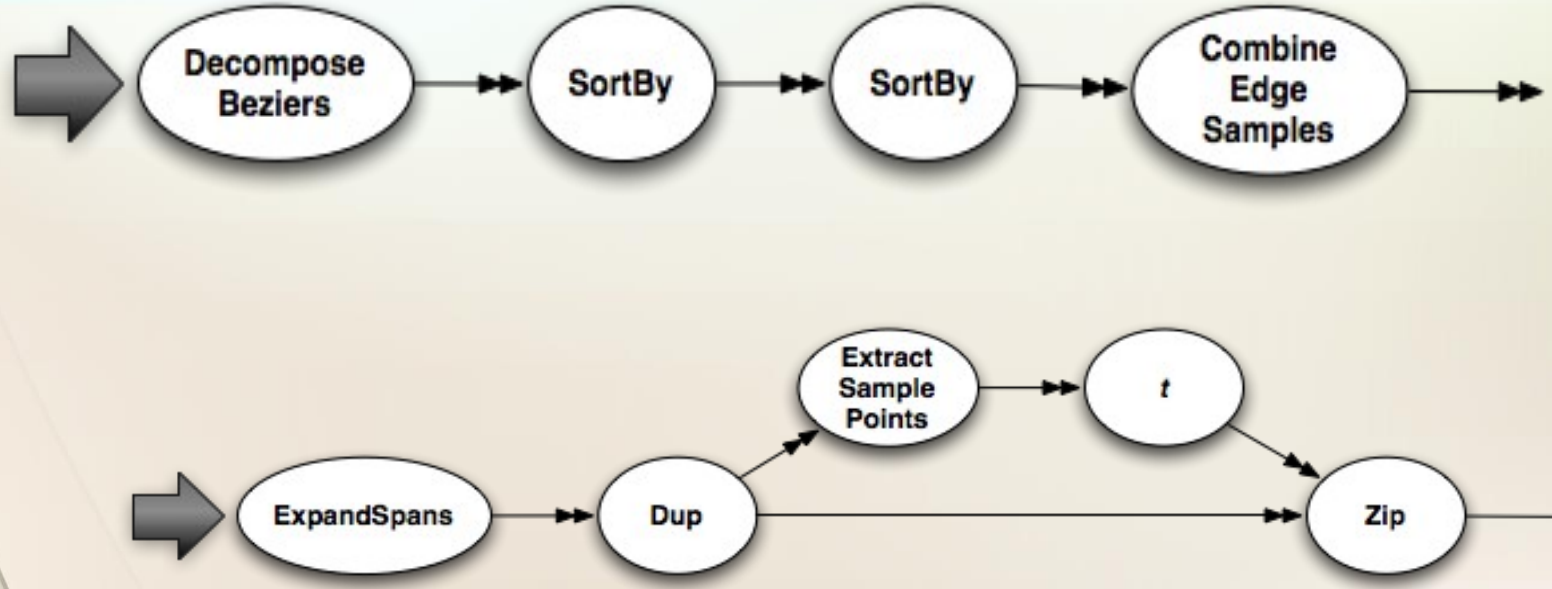
```
type Color         = (a, r, g, b : Real)
type Point         = (x, y : Real)
type Matrix        = (a, b, c, d, e, f : Real)
type Bezier        = (A, B, C : Point)
type EdgeSpan      = (x, y, c, l : Real)
type EdgeSample    = (x, y, a, h : Real)
type PixelCoverage = (x, y, c, ic : Real)

type Texture    = Point >> Color
type Compositor = (Color, Color) >> Color

| (a : Real) | : Real
    { -a if a < 0, a }

(a : Real) ◁ (b : Real) : Real
    {a if a < b, b}

(a : Real) ▷ (b : Real) : Real
    {a if a > b, b}

(a : Real) ~ (b : Real) : Real
    (a + b) / 2

(M : Matrix) ⊗ (A : Point) : Point
    (M.a × A.x + M.c × A.y + M.e, M.b × A.x + M.d × A.y + M.f)

TransformBeziers (M : Matrix) : Bezier >> Bezier
    ∀ (A, B, C)
        >> (M ⊗ A, M ⊗ B, M ⊗ C)

UniformColor (C : Color) : Texture
    ∀ _
        >> (C.a, C.a × C.r, C.a × C.g, C.a × C.b)

CompositeOver : Compositor
    ∀ (a, b)
        >> a + b × (1 - a.a)

ClipBeziers (min, max : Point) : Bezier >> Bezier
    ∀ (A, B, C)
        bmin    = A ◁ B ◁ C
        bmax    = A ▷ B ▷ C
        inside  = min  ≤ bmin ∧ bmax ≤ max
        outside = bmax ≤ min  ∨ max  ≤ bmin
        if inside.x ∧ inside.y
            >> (A, B, C)
        else if outside.x ∨ outside.y
            cA = min ▷ A ◁ max
            cC = min ▷ C ◁ max
            >> (cA, cA ~ cC, cC)
        else
            ABBC    = (A ~ B) ~ (B ~ C)
            nearmin = | ABBC - min | < 0.1
            nearmax = | ABBC - max | < 0.1
            M       = {min if nearmin, max if nearmax, ABBC}
            << (M, B ~ C, C) << (A, A ~ B, M)

CompositeTextures (t1 : Texture, t2 : Texture, c : Compositor) : Texture
    ⇒ DupZip (t1, t2) → c

ExpandSpans : EdgeSpan >> PixelCoverage
    ∀ (x, y, c, l)
        if c
            >> (x, y, c, 1 - c)
        if l > 0
            << (x + 1, y, 1, l - 1)

ExtractSamplePoints : PixelCoverage >> Point
    ∀ (x, y, _, _)
        >> (x, y)

ApplyTexture (t : Texture) : EdgeSpan >> (Color, PixelCoverage)
    ⇒ ExpandSpans → DupZip (ExtractSamplePoints → t, (→))

DecomposeBeziers : Bezier >> EdgeSample
    ∀ (A, B, C)
        inside = (⌊ A ⌋ = ⌊ C ⌋ ∨ ⌈ A ⌉ = ⌈ C ⌉)
        if inside.x ∧ inside.y
            P = ⌊ A ⌋ ◁ ⌊ C ⌋
            w = P.x + 1 - (C.x ~ A.x)
            h = C.y - A.y
            >> (P.x + 1/2, P.y + 1/2, w × h, h)
        else
            ABBC    = (A ~ B) ~ (B ~ C)
            min     = ⌊ ABBC ⌋
            max     = ⌈ ABBC ⌉
            nearmin = | ABBC - min | < 0.1
            nearmax = | ABBC - max | < 0.1
            M       = {min if nearmin, max if nearmax, ABBC}
            << (M, B ~ C, C) << (A, A ~ B, M)

CombineEdgeSamples : EdgeSample >> EdgeSpan
    (x, y, A, H) = 0
    ∀ (x', y', a, h)
        if y' = y
            if x' = x
                A' = A + a
                H' = H + h
            else
                l = {x' - x - 1 if |H| > 0.5, 0}
                >> (x, y, |A| ◁ 1, l)
                A' = H + a
                H' = H + h
        else
            >> (x, y, |A| ◁ 1, 0)
            A' = a
            H' = h
    >> (x, y, |A| ◁ 1, 0)

Rasterize : Bezier >> EdgeSpan
    ⇒ DecomposeBeziers → SortBy (@x) → SortBy (@y) → CombineEdgeSamples
```

# Talk Overview

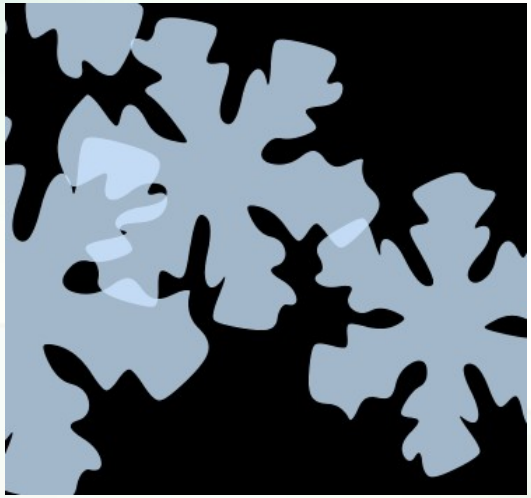
- The Nile language
  - Dataflow
  - Data manipulation
- Graphics rendering in Nile
- Parallelism in Nile
- The “Frankenstein” Application
- Future Work
- Related Work

# Nile Dataflow

- Processes communicate asynchronously via unidirectional, homogenous, typed streams of structured data
- Single input / single output (with some exceptions)



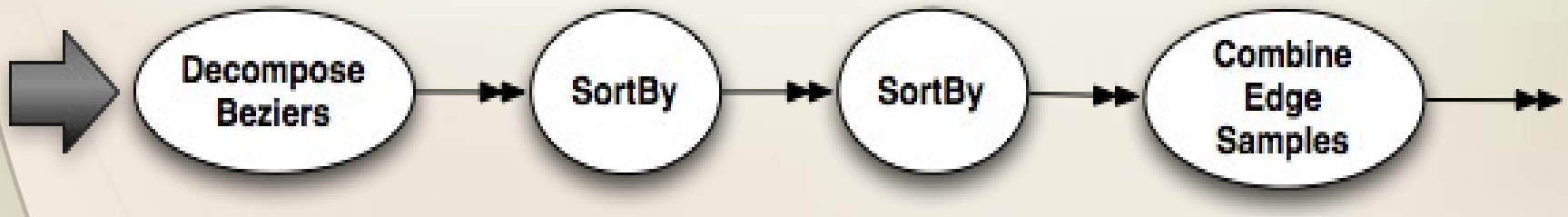
# Rendering Pipeline



```
TransformBeziers (matrix) →
      ClipBeziers (min, max) →
            Rasterize →
                  ApplyTexture (texture) →
                        WriteToImage (image)
```

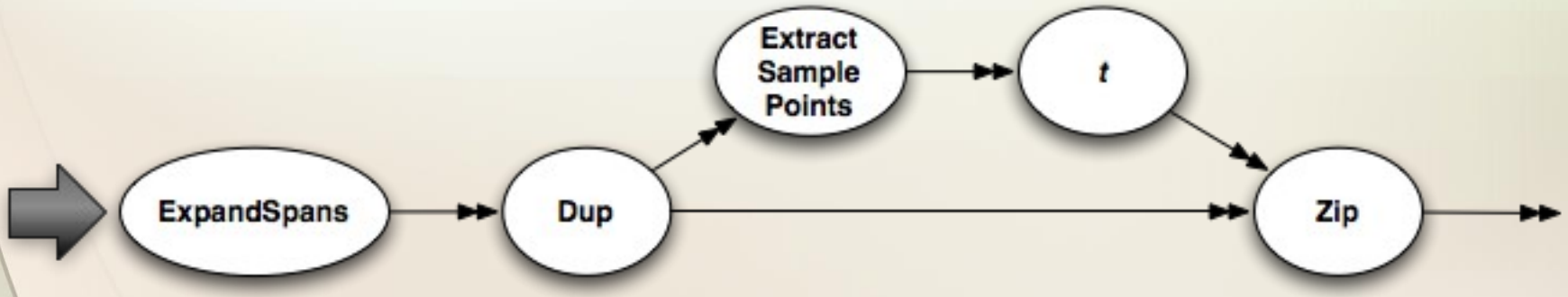
# Rasterization Pipeline

```
Rasterize : Bezier >> EdgeSpan
      ⇒ DecomposeBeziers →
        SortBy (@x) → SortBy (@y) →
        CombineEdgeSamples
```



# Texturing Pipeline

```
ApplyTexture (t : Texture) : EdgeSpan >> (Color, PixelCoverage)
   ⇒ ExpandSpans →
     DupZip (ExtractSamplePoints → t, (→))
```



# Data Manipulation

- Statically typed with type inference
- Single assignment
- User-defined record types and operators
- Pattern matching
- Syntax for stream I/O

# Data Manipulation

```
Sum : Real >> Real
    s = 0
    ∀ x
        s' = s + x
        >> s
     >> s
```

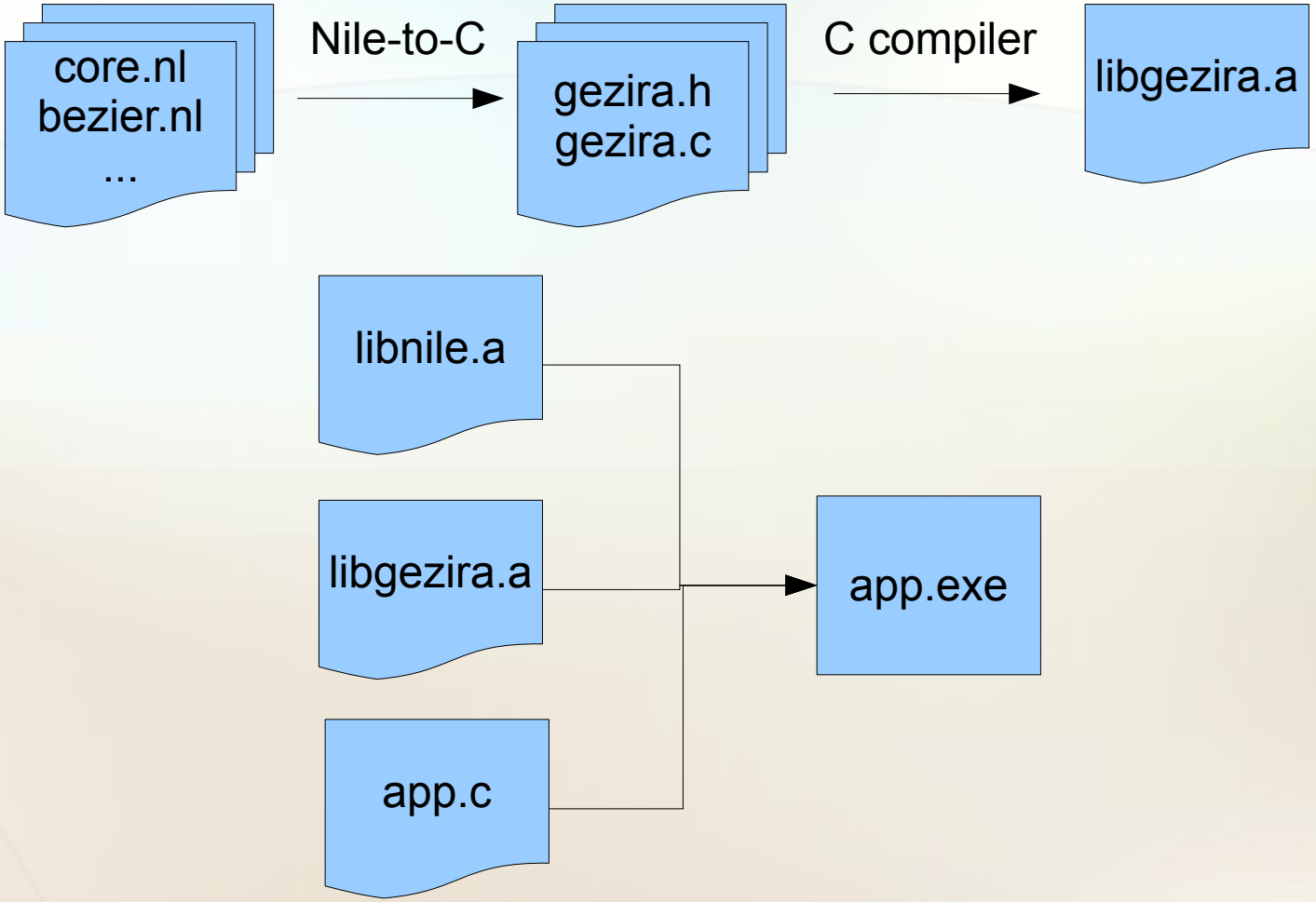
# User-defined Types and Operators

```
type Point  = (x, y : Real)
type Matrix = (a, b, c, d, e, f : Real)
type Bezier = (A, B, C : Point)

(M : Matrix) × (A : Point) : Point
    (M.a × A.x + M.c × A.y + M.e,
     M.b × A.x + M.d × A.y + M.f)

TransformBeziers (M : Matrix) : Bezier >> Bezier
    ∀ (A, B, C)
        >> (M × A, M × B, M × C)
```

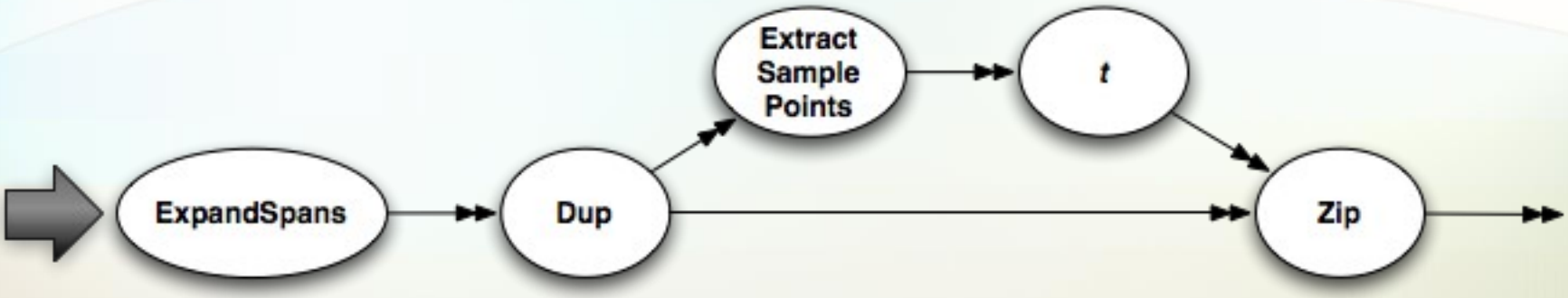
# Gezira: A 2D Vector Graphics Renderer Written in Nile

- In under 400 lines of code:
  - Anti-aliased rasterization of Bezier shapes
  - Affine transformation
  - Geometry clipping
  - 26 compositing operators
  - Texture transformation and extend styles
  - Bilinear and bicubic filters
  - Gaussian blur
  - Multi-stop linear and radial color gradients
  - Pen stroke paths with 3 join styles and 3 cap styles
  - Geometry bounds calculation

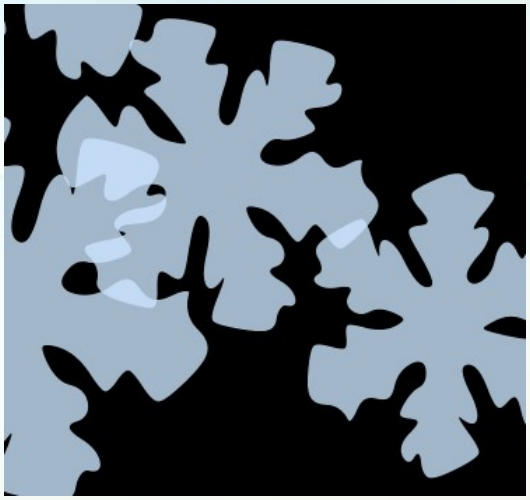
# Workflow

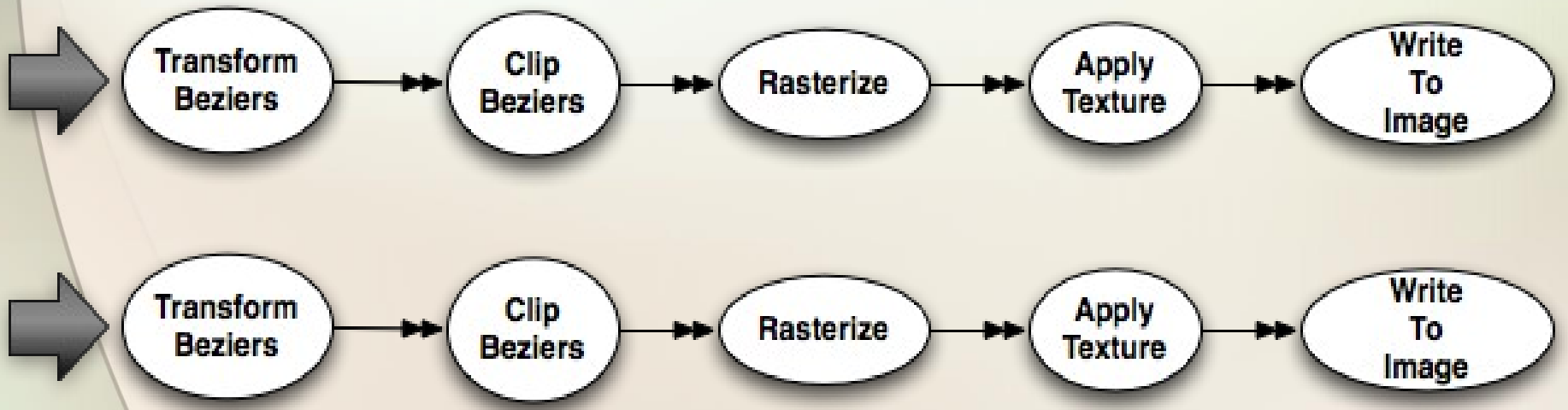

core.nl
bezier.nl
...
Nile-to-C
gezira.h
gezira.c
C compiler
libgezira.a
libnile.a
libgezira.a
app.c
app.exe

# Intra-pipeline Parallelism



# Inter-pipeline Parallelism

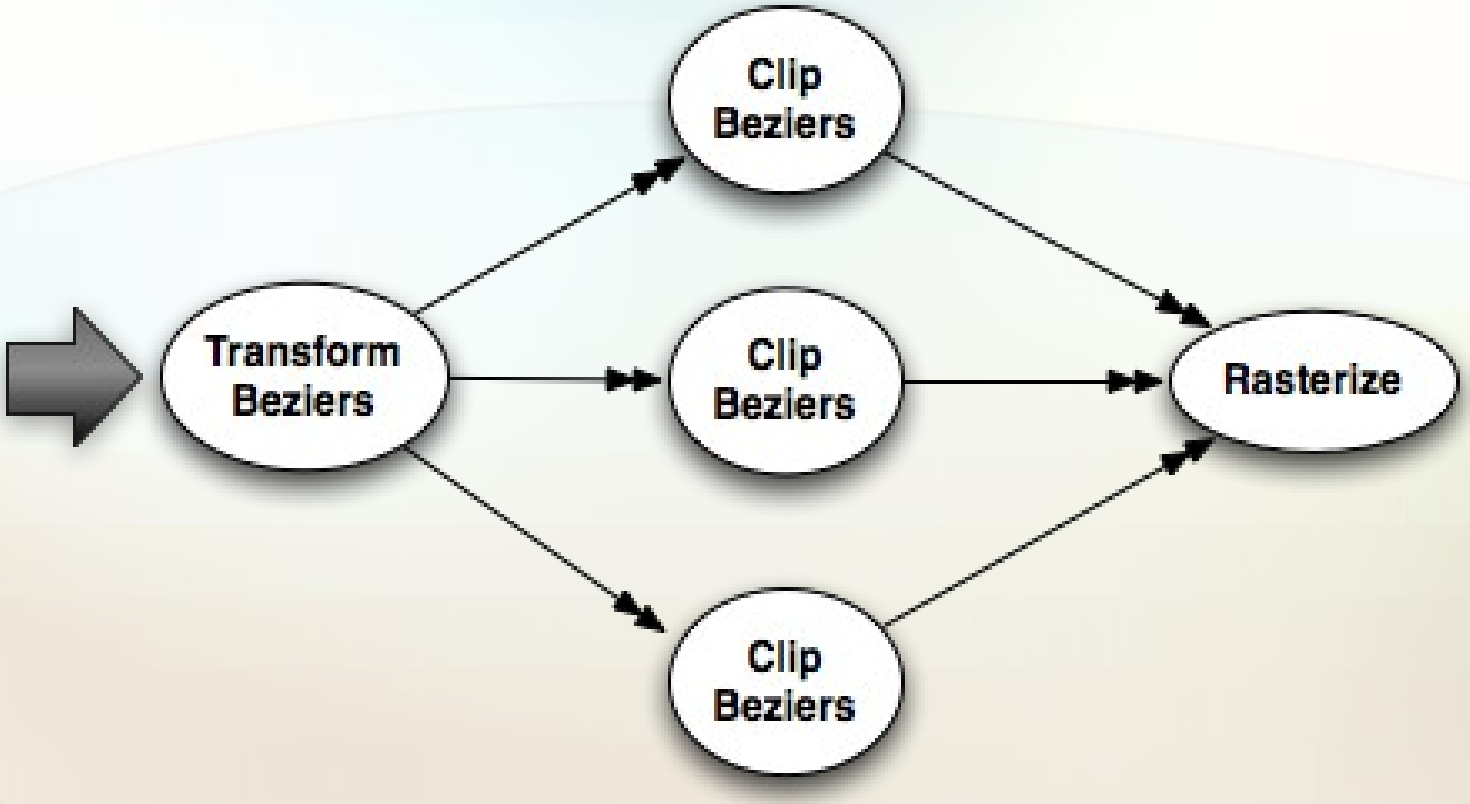




# Instruction-level SIMD Parallelism

```
(M : Matrix) × (A : Point) : Point
    (M.a × A.x + M.c × A.y + M.e,
     M.b × A.x + M.d × A.y + M.f)

TransformBeziers (M : Matrix) : Bezier >> Bezier
    ∀ (A, B, C)
        >> (M × A, M × B, M × C)
```

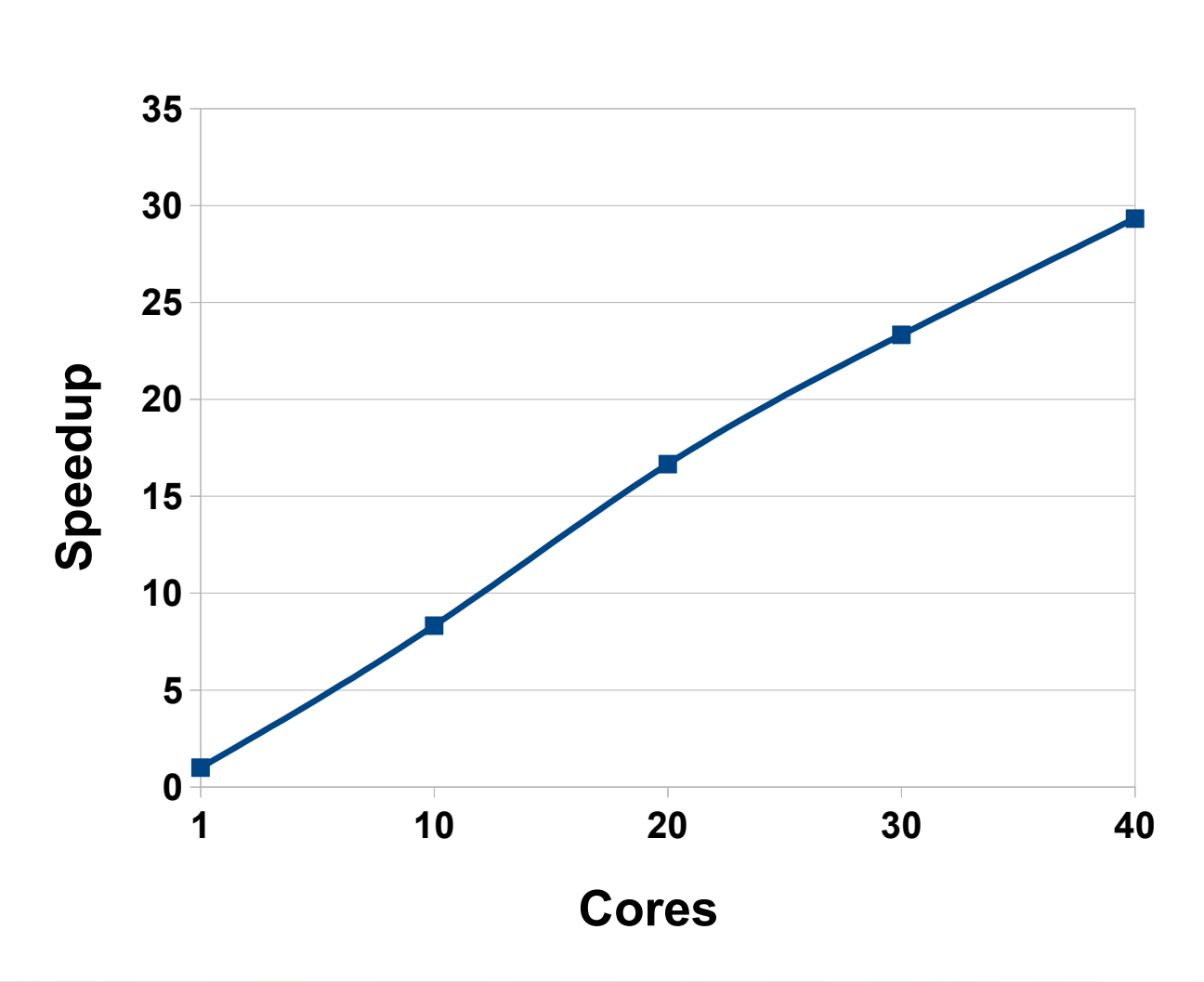
# Process-level Data Parallelism



# Nile Runtime

- Multi-threaded
- Load balancing
- Heap balancing
- User space synchronization

# Speedup on 40 core machine



# Meet “Frank”

# Future Work

- More applications
  - Data compression (zlib, png)
  - Audio/video decoding
  - 3D graphics
  - Image processing
- More compiler backends
  - OpenCL (for GPUs)
  - Javascript
- More language features
  - Sliding windows
  - Feedback networks
  - Binary streams

# Related Work

- Programming models
  - Kahn Process Networks [Kahn74, Kahn76]
  - Dataflow Process Networks [Lee95]
- Programming languages
  - Lucid [Wadge85]
  - VAL [Acherman79], later SISAL [Mcgraw85]
  - Id [Arvind90], later Parallel Haskell (pH) [Nikhil93]
  - Streamit [Thies02]
  - Single Assignment C [Scholz03] and S-Net [Grelck07]

# Extra slides...

# Model of Graphics Rendering?

From Computer Graphics: Principles and Practice [Foley et al.]:

Once the ET has been formed, the following processing steps for the scan-line algorithm are completed:

1. Set y to the smallest y coordinate that has an entry in the ET, that is, y for the first nonempty bucket.
2. Initialize the AET to be empty.
3. Repeat until the AET and ET are empty;
   (a) Move from ET bucket y to the AET edges whose ymin = y (add entering edges).
   (b) Remove from the AET those entries for which y = ymax (edges not involved in the next scan line), then sort the AET on x.
   (c) Fill in desired pixel values on scan line y by using pairs of x coordinates from the AET (suitably rounded).
   (d) Increment y by 1 (to the coordinate of the next scan line).
   (e) For each nonvertical edge remaining in the AET, update x for the new y.

Given the x and y coordinates of the lower-left corner of a pixel, the coverage contribution of an edge $\overrightarrow{AB}$ can be calculated as follows:

$$\sigma(P,Q) = (Q_y - P_y)(x + 1 - \frac{Q_x + P_x}{2})$$

$$\begin{aligned} \gamma(P) &= min(x+1, max(x, P_x)), \\ & \quad min(y+1, max(y, P_y)) \end{aligned}$$

$$\begin{aligned} \omega(P) &= \frac{1}{m}(\gamma(P)_y - P_y) + P_x, \\ & \quad m(\gamma(P)_x - P_x) + P_y \end{aligned}$$

$$\begin{aligned} coverage(\overrightarrow{AB}) &= \sigma(\gamma(A), \gamma(\omega(A))) + \\ & \quad \sigma(\gamma(\omega(A)), \gamma(\omega(B))) + \\ & \quad \sigma(\gamma(\omega(B)), \gamma(B)) \end{aligned}$$

The total coverage contribution of a polygon is the linear combination of the edge contributions, with some additional adjustment:

$$min(|\sum coverage(\overrightarrow{AB}_i)|, 1)$$

# TODO: size graph of cairo vs. gezira