Supplemental – Picture: A probabilistic programming language for scene perception

Tejas D Kulkarni
MIT
tejask@mit.edu

Pushmeet Kohli
MSR Cambridge
pkohli@microsoft.com

Joshua B Tenenbaum
MIT
jbt@mit.edu

Vikash Mansinghka
MIT
vkm@mit.edu

1. 3D medially-symmetric object reconstruction program

As illustrated in Figure 1, the height H of the object is sampled from a uniform distribution. 3D Objects can consist of several sub-parts. Without the loss of generality and for simplicity, we study objects with up-to two sub-parts and circular cross-section. We sample a cut C along the medial axis of the object using the beta distribution, resulting in two independent GPs spanning the cut proportions. Since the smoothness and profile of 3D objects is a priori unknown, we need to do hyper-parameter inference on the bandwidths L_1 and L_2 of the covariance kernel of the GPs. The resulting points $f_{\mathcal{GP}}(x)$ from GPs are passed to the graphics simulator for lathing based mesh generation, which results in the generation of I_R. During inference, reconstructing 3D objects amounts to calculating the posterior $P(S = \{H, C, L_1, L_2\}|I_D)$. While collecting results, we found that running multiple indepedent chains and aggregating the estimates (MAP or average) gave much better parses. For visualizing the underlying stochastic process, refer to supplemental video.

The 3D shape program in Figure 3 could then be roughly formalized as follows:

$$H \sim Uniform(a_0, b_0) \text{ and } C \sim a_c + b_c * Beta(1, H)$$

$$L_1 \sim a_1 + b_1 Beta(2, 5), x_1 = [a_0, C]$$

$$L_2 \sim a_1 + b_1 Beta(2, 5), x_2 = [C + 1, b_0]$$

$$k(x_p^i, x_p^j) = \exp\Big(-\frac{(x_p^i - x_p^j)}{2L_1^2}\Big)$$

where $p \in \{1, 2\}$ denote parts and

$$\min(x_p) \leq (i, j) \leq \max(x_p)$$

$$f^p_{\mathcal{GP}}(x) = \begin{cases} \text{normalize}(\mathcal{GP}(0, k(x_p^i, x_p^{i'}))) & 0 \leq \text{length}(x_p) \\ 0 & \text{otherwise} \end{cases}$$

$$I_R = g_{LATHE}(\{f^1_{\mathcal{GP}}(x_1^m)\}, \{f^2_{\mathcal{GP}}(x_2^n)\})$$

where $min(x_1) \leq m \leq max(x_1)$ and $min(x_2) \leq n \leq max(x_2)$.

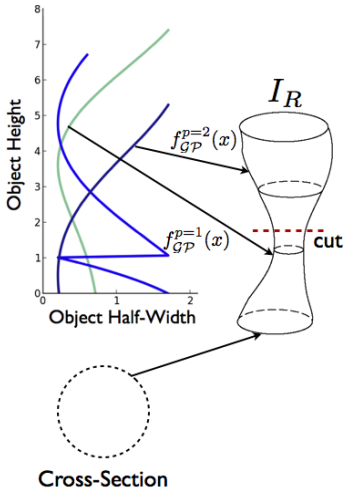


Figure 1: **3D Shape Program Visualization**

2. 3D Human Pose Program

It is important to note that this particular program faces a lot of difficulty under clutter mainly due to lack of robust bottom-up features. Moreover, the graphics program could be significantly improved by using BlendSCAPE[2] model along with approximately reasoning about people's clothes for more accurate body-part localization. Please refer to Figure 4 to view the probabilistic program. While collecting results, we found that running multiple indepedent chains and aggregating the estimates (MAP or average) gave much better parses.

3. Generative Face Program

Since the dimensionality of the face program is high (8 sets of 100 dimensional continuous coupled latent variables), single-site metropolis hastings algorithm is highly inefficient as the number of simulation updates scale linearly with dimensionality of latents. Picture allows us to easily swap inference scheme, which enabled us to quickly discover that elliptical slice moves are significantly more

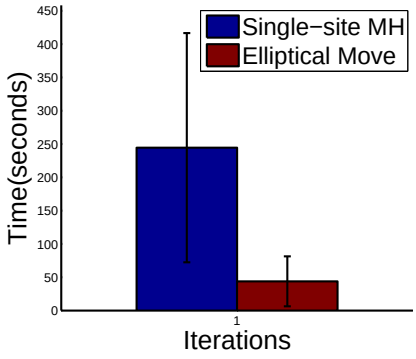


Figure 2: **Inference run-time comparison for face program:** We ran 30 independent inference runs each by toggling the inference scheme between single-site metropolis hastings and elliptical slice proposals. Elliptical moves give significant speedup to reach a certain level of score, which is expected as single-site updates will scale linearly with dimensionality of latents.

efficient than Metropolis-Hastings proposals (see Figure 2). While collecting results, we found that running multiple indepedent chains and aggregating the estimates (MAP or average) gave much better parses.

4. Mesh Induction: Differentiable Rendering Program

See Figure 5a for an example program where the task is to do parameter estimation of camera and light variables given an observed image. Picture allows users to easily switch between using sampling schemes for a given program (*infer(callback, iterations, scheme)*, where *scheme* could be MCMC or HMC).

References

[1] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam,

[2] D. A. Hirshberg, M. Loper, E. Rachlin, and M. J. Black. Coregistration: Simultaneous alignment and modeling of articulated 3d shape. In *ECCV*. 2012.

[3] M. M. Loper and M. J. Black. Opendr: An approximate differentiable renderer. In *ECCV 2014*. 2014.

[4] R. Neal. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2, 2011.

```
function GP(xs,L)
  cov = zeros(length(xs),length(xs))
  mu = zeros(length(xs))
  for i=1:length(xs)
    for j=1:length(xs)
      cov = exp(-(xs[i]-xs[j])^2/(2*l^2))
    end
  end
  return mu,cov
end

function PROGRAM(observation_dt)
  RES = 120
  height = Uniform(5,7.8,1,1)
  xs = [0:height/RES:height]
  samples = zeros(length(xs))

  cut_var = Beta(1,5,1,1)
  cut1,cut2 = sample_cut(xs,height,cut_var,RES)
  l1 = 5*Beta(2,5,1,1)
  l2 = 5*Beta(2,5,1,1)

  if length(cut1)>0
    mu,cov = GP(xs[cut1],l1)
    samples[cut1]=MvNormal(mu, cov)
    samples[cut1] = normalize(samples[cut1])
  end

  if length(cut2)>0
    mu,cov = GP(xs[cut2],l2)
    samples[cut2]=MvNormal(mu, cov)
    samples[cut2] = normalize(samples[cut2])
  end
  #camera:[scale,rotation,translation]
  camera = [Normal(0.98,0.1,1,3), Normal(0,5,1,3),
            Uniform(-1,1,1,3)]
  #Call Blender API
  render = render("profile", samples, "cross-sec", xs, "camera", camera)
  edgemap = canny(rendering,1.0);
  valid_indxs = np.where(edgemap>0)
  D = np.multiply(observation_distance_transform, rendering)
  observe(0,Normal(0,0.35),D)
end

global observation_distance_transform
edge_img = canny(imread("test.png"),1.0)
observation_distance_transform = scipy.distance_transform_bf(edge_img)

TR = trace(PROGRAM,[])
infer(TR, debug_callback,100,"MCMC")
```

Figure 3: **Picture code for 3D Object Reconstruction via Lathing:** Gaussian Process based 3D reconstruction program of lathe objects. This program samples 3D shapes with two independent sub-parts. We used probabilistic chamfer distance as the stochastic comparator.

```
function render(hip_location,...,camera)
 args = [hip_location,...,camera]
 blender.println("cmd:skin-modifier")
 rendered_image = blender.println(args)
 return rendered_image
end

function PROGRAM()
 sigma_0 = Uniform(10,50,1,1)
 hip_location = Uniform(-0.35,0,1,3)
 elbowR_rotation = Normal(0,sigma_0,1,3); elbowR_location = Uniform(-1,1,1,3)
 elbowL_rotation = Normal(0,sigma_0,1,3); elbowL_location = Uniform(-1,1,1,3)
 heelL_location = Uniform(-0.1,0.45,1,3); heelR_location = Uniform(-0.1,0.45,1,3)

 #camera:[scale, rotation, translation]
 camera = [Normal(0.98,0.1,1,1), Normal(0,5,1,3), Uniform(-1,1,1,2)]

 #Call Blender API
 rendering = render("bone-id0", hip_location, ... ,...,"camera", camera)

 edgemap = canny(rendering,1.0);
 valid_indxs = np.where(edgemap>0)
 D = np.multiply(observation_distance_transform[valid_indxs], rendering[valid_indxs])
 observe(0,Normal(0,0.35),D)
end

global observation_distance_transform
edge_img = canny(imread("test.png"),1.0)
observation_distance_transform = scipy.distance_transform_bf(edge_img)

TR = trace(PROGRAM,[])
infer(TR, debug_callback,100,"MCMC")
```

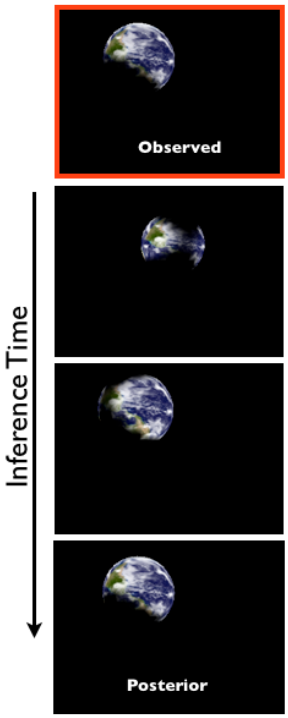
Figure 4: **Picture code for 3D Human Pose:** This program use an existing base mesh of a human body, defines priors over bone location and joints, and enables armature skin-modifier[1] via Picture's Blender engine API. We used probabilistic chamfer distance as the comparator.

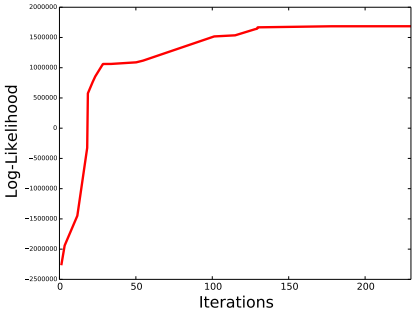
```
function render(TR,ROT,SPH)
  m=util_tests.get_earthmesh()
  V=ch.array(m["v"])
  A = odr.SphericalHarmonics(vn=odr.VertNormals(v=V,
          f= m["f"]),
          components=SPH,
          light_color=ch.ones(3))
  U = odr.ProjectPoints(v=V, f=[300,300.],
          c=[w/2.,h/2.],
          k=ch.zeros(5),
          t=ch.zeros(3), rt=ch.zeros(3))
  ren = odr.TexturedRenderer(vc=A, camera=U,
          f=m["f"],
          bgcolor=[0.,0.,0.],texture_image=m["texture_image"],
          vt=m["vt"], ft=m["ft"],
          frustum={"width"=>w, "height"=>h,"near"=>1,"far"=>20},
          w=w,h=h))
  ren["v"]=TR + V.dot(Rodrigues(ROT))
  return m,V,A,U,ren
end

function PROGRAM(test_image)
  translation = Uniform(-1,1,1,3)
  rotation = Normal(0,10,1,3)
  spherical_harmonics = Normal(0,3,1,9)
  rendering = render(translation, rotation, spherical_harmonics)
  observe_gpyramid(rendering, test_image)
end

global test_image = imread("test.png");
TR = trace(PROGRAM,[])
# SCHEME could potentially be "HMC", "MCMC"
# infer can also be sequentially called with
 combinations of SCHEME's
infer(TR, debug_callback,100,SCHEME)
```

(a) Gradient Based Mesh Program. The task is to optimize or sample the values of translation, rotation and light variables (spherical harmonic co-efficients) given test image. The observation model (or cost function in case of optimization) is a noisy Gaussian pyramid difference between the rendered and observed images.





(b) Typical inference trajectory of the program given an observed image

Figure 5: Example Picture program with automatic gradients for HMC proposals.