Abstract

Recent progress on probabilistic modeling and statistical learning, coupled with the availability of large training datasets, has led to remarkable progress in computer vision. Generative probabilistic models, or “analysis-by-synthesis” approaches, can capture rich scene structure but have been less widely applied than their discriminative counterparts, as they often require considerable problem-specific engineering in modeling and inference, and inference is typically seen as requiring slow, hypothesize-and-test Monte Carlo methods. Here we present Picture, a probabilistic programming language for scene understanding that allows researchers to express complex generative vision models, while automatically solving them using fast general-purpose inference machinery. Picture provides a stochastic scene language that can express generative models for arbitrary 2D/3D scenes, as well as a hierarchy of representation layers for comparing scene hypotheses with observed images by matching not simply pixels, but also more abstract features (e.g., contours, deep neural network activations). Inference can flexibly integrate advanced Monte Carlo strategies with fast bottom-up data-driven methods. Thus both representations and inference strategies can build directly on progress in discriminatively trained systems to make generative vision more robust and efficient. We use Picture to write programs for 3D face analysis, 3D human pose estimation, and 3D object reconstruction – each competitive with specially engineered baselines.

1. Introduction

Probabilistic scene understanding systems aim to produce high-probability descriptions of scenes conditioned on observed images or videos, typically either via discriminatively trained models or generative models in an “analysis by synthesis” framework. Discriminative approaches lend themselves to fast, bottom-up inference methods and relatively knowledge-free, data-intensive training regimes, and have been remarkably successful on many recognition problems \[9, 23, 26, 30\]. Generative approaches hold out the promise of analyzing complex scenes more richly and flexibly \[11, 12, 14, 19, 29, 31, 16, 21\], but have been less widely embraced for two main reasons: Inference typically depends on slower forms of approximate inference, and both model-building and inference can involve considerable problem-specific engineering to obtain robust and reliable results. These factors make it difficult to develop simple variations on state-of-the-art models, to thoroughly explore the many possible combinations of modeling, representation, and inference strategies, or to richly integrate complementary discriminative and generative modeling approaches to the same problem. More generally, to handle increasingly realistic scenes, generative approaches will have to scale not just with respect to data size but also with respect to model and scene complexity. This scaling will arguably require general-purpose frameworks to compose, extend and automatically perform inference in complex structured generative models – tools that for the most part do not yet exist.

Here we present Picture, a probabilistic programming language that aims to provide a common representation language and inference engine suitable for a broad class of generative scene perception problems. We see probabilistic programming as key to realizing the promise of “vision as inverse graphics”. Generative models can be represented via stochastic code that samples hypothesized scenes and generates images given those scenes. Rich deterministic and stochastic data structures can express complex 3D scenes that are difficult to manually specify. Multiple representation and inference strategies are specifically designed to address the main perceived limitations of generative approaches to vision. Instead of requiring photo-realistic generative models with pixel-level matching to images, we can compare hypothesized scenes to observations using a hierarchy of more abstract image representations such as contours, discriminatively trained part-based skeletons, or deep neural network features. Available Markov Chain Monte Carlo (MCMC) inference algorithms include not only traditional Metropolis-Hastings, but also more advanced techniques for inference in high-dimensional continuous spaces, such as elliptical slice sampling, and Hamiltonian Monte Carlo which can exploit the gradients of automatically differentiable renderers. These top-down inference approaches are integrated with bottom-up and automatically constructed data-driven
Figure 1: Overview: (a) All models share a common template; only the scene description \( S \) and image \( I_D \) changes across problems. Every probabilistic graphics program \( f \) defines a stochastic procedure that generates both a scene description and all the other information needed to render an approximation \( I_R \) of a given observed image \( I_D \). The program \( f \) induces a joint probability distribution on these program traces \( \rho \). Every \textit{Picture} program has the following components. \textbf{Scene Language}: Describes 2D/3D scenes and generates particular scene related trace variables \( S^p \in \rho \) during execution. \textbf{Approximate Renderer}: Produces graphics rendering \( I_R \) given \( S^p \) and latents \( X^r \) for controlling the fidelity or tolerance of rendering. \textbf{Representation Layer}: Transforms \( I_D \) or \( I_R \) into a hierarchy of coarse-to-fine image representations \( \nu(I_D) \) and \( \nu(I_R) \) (deep neural networks \cite{23,23}, contours \cite{29} and pixels). \textbf{Comparator}: During inference, \( I_R \) and \( I_D \) can be compared using a likelihood function or a distance metric \( \lambda \) (as in Approximate Bayesian Computation \cite{44}). (b) \textbf{Inference Engine}: Automatically produces a variety of proposals and iteratively evolves the scene hypothesis \( S \) to reach a high probability state given \( I_D \). (c): Representative random scenes drawn from probabilistic graphics programs for faces, objects, and bodies.

proposals, which can dramatically accelerate inference by eliminating most of the “burn in” time of traditional samplers and enabling rapid mode-switching.

We demonstrate \textit{Picture} on three challenging vision problems: inferring the 3D shape and detailed appearance of faces, the 3D pose of articulated human bodies, and the 3D shape of medially-symmetric objects. The vast majority of code for image modeling and inference is reusable across these and many other tasks. We shows that \textit{Picture} yields performance competitive with optimized baselines on each of these benchmark tasks.

2. \textbf{Picture Language}

\textit{Picture} descends from our earlier work on generative probabilistic graphics programming (GPGP) \cite{31}, and also incorporates insights for inference from the Helmholtz machine \cite{17,6} and recent work on differentiable renderers \cite{29}. 

```python
function PROGRAM(MU, PC, EV, VERTEX_ORDER)
    # Scene Language: Stochastic Scene Gen
    face=Dict(); shape = []; texture = [];
    for s in ["shape", "texture"]
        for p in ["nose", "eyes", "outline", "lips"]
            coeff = MvNormal(0,1,1,99)
            face[s][p] = MU[s][p] + PC[s][p]*((coeff.*EV[s][p]))
    end
    # Approximate Renderer
    ren_ftrs = getFeatures("CNN_Conv6", rendered_img)
    # Comparator
    using Pixel as Summary Statistics
    obs_img = imread("test.png")
    global obs_img = imread("test.png")
    # Using CNN last conv layer as Summary Statistics
    obs = getFeatures("CNN_Conv6", img)
    # Data-Driven Learning
    learn_datadriven_proposals(TR, 100000, "CNN_Conv6")
    # Inference
    infer(TR,CB,20, ["DATA-DRIVEN"])
    infer(TR,CB,200, ["ELLIPTIICAL"])```

Figure 2: \textit{Picture} code illustration for 3D face analysis: Modules from Figure 1.b are highlighted in bold. Running the program unconditionally (by removing \textit{observe}'s in code) produces random faces as shown in Figure 1. Running the program conditionally (keeping \textit{observe}'s) on \( I_D \) results in posterior inference as shown in Figure 3. The variables \( MU, PC, EV \) correspond to the mean shape/texture face, principal components, and eigenvectors respectively (see \cite{39} for details). These arguments parametrize the prior on the learned shape and appearance of 3D faces. The argument \textbf{VERTEX_ORDER} denotes the ordered list of vertices to render triangle based meshes. The \textit{observe} directive constrains the program execution based on both the pixel data and CNN features. The \textit{infer} directive starts the inference engine with the specified set of inference schemes (takes the program trace, a callback function \textbf{CB} for debugging, number of iterations and inference schemes). In this example, data-driven proposals are run for a few iterations to initialize the sampler, followed by slice sampling moves to further refine the high dimensional scene latents.
and informed samplers \cite{19}. GPGP aimed to address the main challenges of generative vision by representing visual scenes as short probabilistic programs with random variables, and using a generic MCMC (single-site Metropolis-Hastings) method for inference. However, due to modeling limitations of earlier probabilistic programming languages, and the inefficiency of the Metropolis-Hastings sampler, GPGP was limited to working with low-dimensional scenes, restricted shapes, and low levels of appearance variability. Moreover, it did not support the integration of bottom-up discriminative models such as deep neural networks \cite{23,25} for data-driven proposal learning. Our current work extends the GPGP framework in all of these directions, letting us tackle a richer set of real-world 3D vision problems.

*Picture* is an imperative programming language, where expressions can take on either deterministic or stochastic values. We use the transformational compilation technique \cite{46} to implement *Picture*, which is a general method of transforming arbitrary programming languages into probabilistic programming languages. Compared to earlier formulations of GPGP, *Picture* is dynamically compiled at run-time (JIT-compilation) instead of interpreting, making program execution much faster.

A *Picture* program $f$ defines a stochastic procedure that generates both a scene description and all other information needed to render an approximation image $I_R$ for comparison with an observed image $I_D$. The program $f$ induces a joint probability distribution on the program trace $\rho = \{\rho_i\}$, the set of all random choices $i$ needed to specify the scene hypothesis $S$ and render $I_R$. Each random choice $\rho_i$ can belong to a familiar parametric or non-parametric family of distributions, such as *Multinomial, MvNormal, DiscreteUniform, Poisson, or Gaussian*, but in being used to specify the trace of a probabilistic graphics program, their effects can be combined much more richly than is typical for random variables in traditional statistical models.

Consider running the program in Figure \ref{fig:program} unconditionally (without observed data): as different $\rho_i$’s are encountered (for e.g. *coeff*), random values are sampled w.r.t their underlying probability distribution and cached in the current state of the inference engine. Program execution outputs an image of a face with random shape, texture, camera and lighting parameters. Given image data $I_D$, inference in *Picture* programs amounts to iteratively sampling or evolving program trace $\rho$ to a high probability state while respecting constraints imposed by the data (Figure \ref{fig:program}). This constrained simulation can be achieved by using the *observe* language construct (see code in Figure \ref{fig:program}), first proposed in Venture \cite{32} and also used in \cite{35,47}.

### 2.1. Architecture

In this section, we will explain the essential architectural components highlighted in Figure \ref{fig:architecture} (see Figure \ref{fig:architecture} for a summary of notation used).

**Scene Language:** The scene language is used to describe 2D/3D visual scenes as probabilistic code. Visual scenes can be built out of several graphics primitives such as: description of 3D objects in the scene (e.g. mesh, z-map, volumetric), one or more lights, textures, and the camera information. It is important to note that scenes expressed as probabilistic code are more general than parametric prior density functions as is typical in generative vision models. The probabilistic programs we demonstrate in this paper embed ideas from computer-aided design (CAD) and non-parametric Bayesian statistics \cite{37} to express variability in 3D shapes.

**Approximate Renderer (AR):** *Picture*’s AR layer takes in a scene representation trace $S^\rho$ and tolerance variables $X^\rho$, and uses general-purpose graphics simulators (Blender\cite{5} and OpenGL) to render 3D scenes. The rendering tolerance $X^\rho$ defines a structured noise process over the rendering and is useful for the following purposes: (a) to make automatic inference more tractable or robust, analogous to simulated annealing (e.g. global or local blur variables in GPGP \cite{31}), and (b) to soak up model mismatch between the true scene rendering $I_D$ and the hypothesized rendering $I_R$. Inspired by the *differentiable renderer* \cite{29}, *Picture* also supports expressing AR’s entire graphics pipeline as *Picture* code, enabling the language to express end-to-end differentiable generative models.

**Representation Layer (RL):** To avoid the need for photorealistic rendering of complex scenes, which can be slow and modeling-intensive, or for pixel-wise comparison of hypothesized scenes and observed images, which can sometimes yield posteriors that are intractable for sampling-based inference, the RL supports comparison of generated and observed images in terms of a hierarchy of abstract features.
Figure 4: Formal Summary: The scene $S$ can be conceptualized as a program that describes the structure of known or unknown number of objects, texture-maps, lighting and other scene variables. The symbol $T$ denotes the number of times the program $f$ is executed to generate data-driven proposals (see section 3.2 for details). The rendering differentiator produces gradients of the program density with respect to continuous variables $S_{real}$ in the program.

The RL can be defined as a function $\nu$ which produces summary statistics given $I_D$ or $I_R$, and may also have internal parameters $\theta_\nu$ (e.g. weights of a deep neural net). For notational convenience, we denote $\nu(I_D; \theta_\nu)$ and $\nu(I_D; \theta_\nu)$ to be $\nu(I_D)$ and $\nu(I_R)$ respectively. RL produces summary statistics (features) that are used in two scenarios: (a) to compare the hypothesis $I_R$ with observed image $I_D$ during inference (RL denoted by $\nu$ in this setting), and (b) as a dimensionality reduction technique for hashing learned data-driven proposals (RL denoted by $\nu_{dd}$ and its parameters $\theta_{eq}u$). Picture supports a variety of summary statistic functions including raw pixels, contours [8] and supervised/unsupervised convolutional neural network (CNN) architectures [23, 25].

Likelihood and Likelihood-free Comparator: Picture supports likelihood $P(I_D|I_R)$ inference in a bayesian setting. However, in the presence of black-box rendering simulators, the likelihood function is often unavailable in closed form. Given an arbitrary distance function $d(\nu(I_D), \nu(I_R))$ (e.g. $L_1$ error), approximate bayesian computation (ABC) [44] can be used to perform likelihood-free inference.

### 3. Inference

We can formulate the task of image interpretation as approximately sampling mostly likely values of $S^0$ given observed image $I_D$ ($L$ stands for $P(I_D|I_R, X^p)$):

$$P(S^0|I_D) \propto \int P(S^0)P(X^p)d_{\text{render}}(S, X^p)(I_R) \; L \; dX^p$$

Automatic inference in Picture programs can be especially hard due to a mix of discrete and continuous scene variables, which may be independent a priori but highly coupled in their posterior distributions (“explaining away”), and also because clutter, occlusion or noise can lead to local maxima of the scene posterior.

Given a program trace $\rho$, probabilistic inference amounts to updating $(S^0, X^p)$ to $(S^p, X^p)$ until convergence via proposal kernels $q((S^0, X^p) \rightarrow q(S^0, X^p))$. Let $K = {[S^0]} + {[X^p]}$ and $K' = {[S^p]} + {[X^p]}$ be the total number of random choices in the execution before and after applying the proposal kernels $q(.)$. Let the log-likelihoods of old and new trace be $L = P(I_D|I_R, X)$ and $L' = P(I_D|I_R', X')$ respectively. Let us denote the probabilities of deleted and newly created random choices created in $S^p$ to be $P(S^p)\delta_{\text{del}}$ and $P(S^p)\delta_{\text{new}}$ respectively. Let $q(S, X^p) \rightarrow q(S^p, X^p)$ and $q(S, X^p) \rightarrow q(S^p, X^p)$. The new trace $(S^p, X^p)$ can now be accepted or rejected using the acceptance ratio:

$$\min \left( 1, \frac{1}{L} \frac{P(S^p)P(X^p)q(S, X^p)q(S^0, X^p)K'P(S^d_{\text{del}})}{L'P(S^p)P(X^p)q(S, X^p)q(S^0, X^p)K'P(S^p_{\text{del}})} \right).$$

### 3.1. Distance Metrics and Likelihood-free Inference

The likelihood function in closed form is often unavailable when integrating top-down automatic inference with bottom-up computational elements. Moreover, this issue is exacerbated when programs use black-box rendering simulators. Approximate bayesian computation (ABC) allows Bayesian inference in likelihood-free settings, where the basic idea is to use a summary statistic function $\nu(\cdot)$, distance metric $\lambda(\nu(I_D), \nu(I_R))$ and tolerance variable $X^p$ to approximately sample the posterior distribution [44].

Inference in likelihood-free settings can also be interpreted as a variant of the probabilistic approximate MCMC algorithm [44], which is similar to MCMC but with an additional tolerance parameter $\epsilon \in X^p$ on the observation model. We can interpret our approach as systematically reasoning about the model error arising due to the difference of generative model’s “cartoon” view of the world with reality. Let $\Theta$ be the space of all possible renderings $I_R$ that could be hypothesized and $P_\nu$ be the error model (e.g. Gaussian). The target stationary distribution that we wish to sample can be expressed as:

$$P(S^0|I_D) \propto \int \ldots P(S^0)P_\nu(\nu(I_D) - \nu(I_R))P(\nu(I_R)|S^0)dI_R.$$
During inference, the updated scene $S'_{\theta}$ (assuming random choices remain unchanged, otherwise add terms relating to addition/deletion of random variables as in equation 1) can then be accepted with probability:

$$
\min \left( 1, \frac{P_S(v(I_D) - v(I_R)) P(S'_{\theta}) P(X'_{\theta}) q(S',X') \rightarrow (S,X)}{P_S(v(I_D) - v(I_R)) P(S') P(X') q(S,X) \rightarrow (S',X')} \right).
$$

3.2. Proposal Kernels

In this section, we will propose a variety of proposal kernels for scaling up Picture to complex 3D scenes.

Local and Blocked Proposals from Prior: Single site metropolis hastings moves on continuous variables and Gibbs moves on discrete variables can be useful in many cases. However, because the latent pose variables for objects in 3D scenes (e.g., positions and orientations) are often highly coupled, our inference library allows users to define arbitrary blocked proposals: $q_{\rho}(S'_{\theta}, X'_{\theta}) = \prod_{\rho \in (S_{\theta}, X_{\theta})} P(\rho'_{\theta})$

Gradient Proposal: Picture inference supports automatic differentiation for a restricted class of programs (where each expression provides output and gradients w.r.t input). Therefore it is straightforward to obtain $\nabla_{S_{\text{real}}} \rho$ using reverse mode automatic differentiation, where $S_{\text{real}} \in S_{\theta}$ denotes all continuous variables. This enables us to automatically construct Hamiltonian Monte Carlo proposals \cite{14, 43}.

Elliptical Slice Proposals: To adaptively propose changes to a large set of latent variables at once, our inference library supports elliptical slice moves with or without adaptive step sizes (see Figure 2 for an example) \cite{41, 32}. For simplicity, assume $S_{\text{real}} \sim \mathcal{N}(0, \Sigma)$. We can generate a new sub-trace $S'_{\text{real}}$ efficiently as follows: $S'_{\text{real}} = \sqrt{1 - \alpha^2 S_{\text{real}} + \alpha \theta}$, where $\theta \sim \mathcal{N}(0, \Sigma)$ and $\alpha \sim \text{Uniform}(-1, 1)$.

Data-driven Proposals: The top-down nature of MCMC inference in generative models can be slow, due to the initial “burn-in” period and the cost of mixing among multiple posterior modes. However, vision problems often lend themselves to much faster bottom-up inference based on data-driven proposals \cite{19, 33}. Arguably the most important inference innovation in Picture is the capacity for automatically constructing data-driven proposals by simple learning methods. Such techniques fall under the broader idea of amortizing or caching inference to improve speed and accuracy \cite{40}. We have explored several approaches generally inspired by the Helmholtz machine \cite{17, 6}, and indeed Helmholtz’s own proposals, including using deep learning to construct bottom-up predictors for all or a subset of the latent scene variables in $\rho$ \cite{49, 25}. Here we focus on a simple and general-purpose memory-based approach (similar in spirit to the informed sampler \cite{19}) that can be summarized as follows: We “imagine” a large set of hypothetical scenes sampled from the generative model, store the imagined latents and corresponding rendered image data in memory, and build a fast bottom-up kernel density estimate proposer that samples variants of stored graphics program traces best matching the observed image data — where these bottom-up “matches” are determined using the same representation layer tools we introduced earlier for comparing top-down rendered and observed images. More formally, we construct data-driven proposals $q_{\text{data}}$ as follows:

1. Specify the number of times $T$ to forward simulate (unconditional runs) the graphics program $f$.
2. Draw $T$ samples from $f$ to create program traces $\rho_i'$ and approximate renderings $I_{\rho_i}$, where $1 \leq i \leq T$.
3. Specify a summary statistic function $\nu_{dd}$ with model parameters $\theta_{\nu_{dd}}$. We can use the same representation layer tools introduced earlier to specify $\nu_{dd}$, subject to the additional constraint that feature dimensionalities should be as small as possible to enable proposal learning and evaluation on massive datasets.
4. Fine-tune parameters $\theta_{\nu_{dd}}$ of the representation layer $\nu_{dd}$ using supervised learning to best predict program traces $\{\rho_i'\}_{i=1}^T$ from corresponding rendered images $\{I_{\rho_i}\}_{i=1}^T$. If labeled data is available for full or partial scene traces $\{S_{\theta}'\}$ corresponding to actual observed images $\{I_D\}$, the parameters $\theta_{\nu_{dd}}$ can also be fine-tuned further to predict these labels. (see deep convolutional inverse graphics network \cite{25} as an alternative $\nu_{dd}$, which works in an weakly supervised setting.)
5. Define a hash function $H : \nu(I_{\rho_i}) \rightarrow h_i$, where $h_i$ denotes the hash value for $\nu(I_{\rho_i})$. For instance, $H$ can be defined in terms of K-nearest neighbors or a Dirichlet Process mixture model. Store triplets $\{\rho_i', \nu(I_{\rho_i}), h_i\}$ in a database $D$.
6. To generate data-driven proposals for an observed image $I_D$ with hash value $h_D$, extract all triplets $\{\rho_j', \nu(I_{\rho_j}), h_j\}_{j=1}^N$ that have hash value equal to $h_D$. We can then estimate the data-driven proposal as:

$$
q_{\text{data}}(S_{\theta} \rightarrow S'_{\theta} | C, I_D) = P_{\text{density}}(\{\rho_j'\}_{j=1}^N),
$$

where $P_{\text{density}}$ is a density estimator such as the multivariate gaussian kernel in \cite{19}.

4. Example Picture Programs

To illustrate how Picture can be applied to a wide variety of 2D and 3D computer vision problems, we present three sample applications to the core vision tasks of 3D body pose estimation, 3D reconstruction of objects and 3D face analysis. Although additional steps could be employed to improve
results for any of these tasks, and there may exist better fine-tuned baselines, our goal here is to show how to solve a broad class of problems efficiently and competitively with task-specific baseline systems, using only minimal problem-specific engineering.

4.1. 3D Analysis of Faces

We obtained a 3D deformable face model trained on laser scanned faces from Paysan et al. [36]. After training with this dataset, the model generates a mean shape mesh and mean texture map, along with principal components and eigenvectors. A new face can be rendered by randomly choosing coefficients for the 3D model and running the program shown in Figure 2. The representation layer $\nu$ in this program used the top convolutional-layer features from the ImageNet CNN model [20] as well as raw pixels. (Even better results can be obtained using the deep convolutional inverse graphics network [25] instead of the CNN.) We evaluated the program on a held-out test set of 2D projected images of 3D laser scanned data (dataset from [36]). We additionally produced a dataset of about 30 images from the held-out set with different viewpoints and lighting conditions. In Figure 3, we show qualitative results of inference runs on the dataset.

During experimentation, we discovered that the number of latent variables is large (8 sets of 100 dimensional continuous coupled variables), elliptical slice moves are significantly more efficient than Metropolis-Hastings proposals (see supplementary Figure 2 for quantitative results). We also found that adding learned data-driven proposals significantly outperforms using only the elliptical slice proposals in terms of both speed and accuracy. We trained the data-driven proposals from around 100k program traces drawn from unconditional runs. The summary statistic function $\nu_{ld}$ used were the top convolutional-layer features from the pre-trained ImageNet CNN model [20]. The conditional proposal density $P_{density}$ was a multivariate kernel density function over cached latents with a Gaussian Kernel (0.01 bandwidth). Figure 5 shows the gains in inference from use of a mixture kernel of these data-driven proposals (0.1 probability) and elliptical slice proposals (0.9 probability), relative to a pure elliptical slice sampler.

Many other academic researchers have used 3D deformable face models in an analysis-by-synthesis based approach [23, 22, 1]. However, Picture is the only system to solve this as well as many other unrelated computer vision problems using a general-purpose system. Moreover, the data-driven proposals and abstract summary statistics (top convolutional-layer activations) allow us to tackle the problem without explicitly using 2D face landmarks as compared to traditional approaches.

4.2. 3D Human Pose Estimation

We developed a Picture program for parsing 3D pose of articulated humans from single images. There has been notable work in model-based approaches [13, 27] for 3D human pose estimation, which served as an inspiration for the program we describe in this section. However, in contrast to Picture, existing approaches typically require custom inference strategies and significant task-specific model engineering. The probabilistic code (see supplementary Figure 4) consists of latent variables denoting bone and joints of an articulated 3D base mesh of a body. In our probabilistic code, we use an existing base mesh of a human body, defined priors over bone location and joints, and enable the armature skin-modifier API via Picture’s Blender engine API. The latent scene $S_o$ in this program can be visualized as a tree with the root node around the center of the mesh, and consists of bone location variables, bone rotation variables and camera parameters. The representation layer $\nu$ in this program uses fine-grained image contours [8] and the comparator is expressed as the probabilistic chamfer distance [41].

We evaluated our program on a dataset of humans performing a variety of poses, which was aggregated from KTH [39] and LabelMe [38] images with significant occlusion in the “person sitting” (around 50 total images). This dataset was chosen to highlight the distinctive value of a graphics model-based approach, emphasizing certain dimensions of task difficulty while minimizing others: While graphics simulators for articulated bodies can represent arbitrarily complex body configurations, they are limited with respect to fine-grained appearance (e.g., skin and clothing), and fast methods for fine-grained contour detection currently work well only in low clutter environments. We initially used only single-site MH proposals, although blocked proposals or HMC can somewhat accelerate inference.

We compared this approach with the discriminatively trained Deformable Parts Model (DPM) for pose estimation [48] (referred as DPM-pose), which is notably a 2D pose model. As shown in Figure 6, images with people sitting and heavy occlusion are very hard for the discriminative
Figure 6: Quantitative and qualitative results for 3D human pose program: Refer to supplementary Figure 4 for the probabilistic program. We quantitatively evaluate the pose program on a dataset collected from various sources such as KTH [39], LabelMe [38] images with significant occlusion in the “person sitting” category and the Internet. On the given dataset, as shown in the error histogram in (a), our model is more accurate on average than just using the DPM based human pose detector [48]. The histogram shows average error for all methods considered over the entire dataset separated over each body part.

model to get right – mainly due to “missing” observation signal – while our model-based approach can handle these reasonably if we constrain the knee parameters to bend only in natural ways in the prior. Most of our model’s failure cases, as shown in Figure 6(b), are in inferring the arm position; this is typically due to noisy and low quality feature maps around the arm area due to its small size.

In order to quantitatively compare results, we project the 3D pose obtained from our model to 2D key-points. As shown in Figure 6a, our system localizes these key-points significantly better than DPM-pose on this dataset. However, DPM-pose is a much faster bottom-up method, and we explored ways to combine its strengths with our model-based approach, by using it as the basis for learning data-driven proposals. We generated around 500k program traces by unconditionally running the body pose program. We used a pre-trained DPM pose model [48] as the function $v_{dd}$, and used a similar density function $P_{density}$ as in the face example. As shown in Figure 7 inference using a mixture kernel of data-driven proposals (0.1 probability) and single-site MH (0.9 probability) consistently outperformed pure top-down MH inference in both speed and accuracy. We see this as representative of many ways that top-down inference in model-based approaches could be profitably combined with fast bottom-up methods like DPM-pose to solve richer scene parsing problems more quickly.

4.3. 3D Shape Program

Lathing and casting is a useful representation to express CAD models and inspires our approach to modeling medially-symmetric 3D objects. It is straightforward to generate random CAD object models using a probabilistic program, as shown in supplementary Figure 3. However, the distribution induced by such a program may be quite complex. Given object boundaries in $B \in \mathbb{R}^2$ space, we can lathe an object by taking a cross section of points (fixed for this program), defining a medial axis for the cross section and sweeping the cross section across the medial axis by continuously perturbing with respect to $B$. Capturing the full range of 3D shape variability in real objects will require a very large space of possible boundaries $B$. To this end, $Picture allows flexible non-parametric priors over object profiles: here we generate $B$ from a Gaussian Process [37] (GP). The probabilistic shape program produces an intermediate mesh of all or part of the 3D object (soft-constrained to be in the middle of the scene), which then gets rendered to an image $I_R$ by a deterministic camera re-projection function. The
representation layer and the comparator used in this program were same as those used for the 3D human pose example. The proposal kernel we used during inference consisted of blocked MCMC proposals on all the coupled continuous variables as described in the supplementary material. (For more details of the program and inference summarized here, refer to supplementary Section 1.)

We evaluate this program on an RGB image dataset of 3D objects with large shape variability. We asked CAD experts to manually generate CAD model fits to these images in Blender, and evaluated our approach in comparison to a state-of-the-art 3D surface reconstruction algorithm from [3](SIRFS). To judge quantitative performance, we calculated two metrics: (a) Z-MAE – Shift-invariant surface mean-squared error and (b) N-MSE – mean-squared error over normals [3]. As shown in Figure 8 inference using our probabilistic shape program has a lower Z-MAE and N-MSE score than SIRFS [3], and we also obtain qualitatively better reconstruction results. However, it is important to note that SIRFS predominantly utilizes only low level shape priors such as piece-wise smoothness, in contrast to the high-level shape priors we assume, and SIRFS solves a more general and harder problem of inferring full intrinsic images (shape, illumination and reflectance). In the future, we hope to combine the best of SIRFS-style approaches and our probabilistic CAD programs to reconstruct rich 3D shape and appearance models for generic object classes, robustly and efficiently.

5. Discussion

There are many promising directions for future research in probabilistic graphics programming. Introducing a dependency tracking mechanism could let us exploit the many conditional independencies in rendering for more efficient parallel inference. Automatic particle-filter based inference schemes [47, 24] could extend the approach to image sequences. Better illumination [52], texture and shading models could let us work with more natural scenes. Procedural graphics techniques [2, 10] would support far more complex object and scene models [50, 14, 7, 15]. Flexible scene generator libraries will be essential in scaling up to the full range of scenes people can interpret.

We are also interested in extending Picture by taking insights from learning based “analysis-by-synthesis” approaches such as transforming auto-encoders [18], capsule networks [42] and deep convolutional inverse graphics network [25]. These models learn an implicit graphics engine in an encoder-decoder style architecture. With probabilistic programming, the space of decoders need not be restricted to neural networks and could consist of arbitrary probabilistic graphics programs with internal parameters.

The recent renewal of interest in inverse graphics approaches to vision has motivated a number of new modeling and inference tools. Each addresses a different facet of the general problem. Earlier formulations of probabilistic graphics programming provided compositional languages for scene modeling and a flexible template for automatic inference. Differentiable renderers make it easier to fine-tune the numerical parameters of high-dimensional scene models. Data-driven proposal schemes suggest a way to rapidly identify plausible scene elements, avoiding the slow burn-in and mixing times of top-down MCMC-based inference in generative models. Deep neural networks, deformable parts models and other discriminative learning methods can be used to automatically construct good representation layers or similarity metrics for comparing hypothesized scenes to observed images. Here we show that by integrating all of these ideas into a single probabilistic language and inference framework, it may be feasible to begin scaling up inverse graphics to a range of real-world vision problems.
6. Acknowledgements

We thank Thomas Vetter for giving us access to the Basel face model. T. Kulkarni was graciously supported by the Leventhal Fellowship. This research was supported by ONR award N000141310333, ARO MURI W911NF-13-1-2012 and the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF-1231216. We would like to thank Ilker Yildrim, Yura Perov, Karthik Rajagopal, Alexey Radul, Peter Battaglia, Alan Rusnak, and three anonymous reviewers for helpful feedback and discussions.

References
